SKROLLI

2013.4
SKROLLI.FI
Tietokonekulttuurin erikoislehti

3D-tulostin – uuden ajan jeesusteippi

Verkkokulttuuri ennen ja nyt

((Näinkin voi (ohjelmoida)))

Oma peli, paras peli?

Avatar ydinaavikolla

Sisällysluettelo

Ville-Matias Heikkilä
päätoimittaja

Voitosta voittoon

Skrollin ensimmäinen ilmestymisvuosi onnistui erinomaisesti ja kaiken todennäköisyyden vastaisesti. Keksisimmekö ensi vuodelle jotain vielä mahdottomampaa?

Kun Skrollia alettiin suunnitella syksyllä 2012, idea tuntui monin tavoin kahjolta. Uusi paperinen tietotekniikkalehti aikana, jolloin tavallisenkin kansan lukutottumukset digitalisoituvat vauhdilla? Ohjelmointijuttuja ja painettua koodia, ollaanko siellä aivan sekaisin? Ketä tuollainen kiinnostaa? Mikä tahansa mediatalo olisi hylännyt idean samalta istumalta.

Mahdoton kuitenkin osoittautui mahdolliseksi, kun Skrollin ensimmäinen numero valmistui keväällä. Ykköstä seurasi kolme muuta numeroa, eikä talkoohenki näyttänyt hiipumisen merkkejä missään vaiheessa. Nelosnumerosta tehtiin peräti 84-sivuinen, koska hyvää julkaistavaa oli niin paljon, eikä kaikki edes mahtunut mukaan. Lisäksi useimmat tekijät kieltäytyivät heille tarjotusta palkkiosta, mikä oli jo aivan uskomatonta.

Jotta Skrolli-projekti pysyisi tuoreena ja kiinnostavana, sen ei kannata tyytyä nykyisiin saavutuksiinsa vaan tavoitella uusia haasteita. Lehti on nyt saavuttanut jonkinlaisen aseman pienen piirin harrastelehtenä, mutta se voisi olla paljon enemmänkin. Tietokonekulttuurin lonkerot ulottuvat kaikkialle yhteiskuntaan, joten Skrollikin voisi olla taho, jonka näkemystä ja asiantuntemusta arvostetaan myös lukijakuntansa ulkopuolella.

Vuonna 2014 lähdemme siis kohti jälleen uutta tuntematonta. Jos haluat olla mukana, käy ihmeessä tilaamassa lehti. Kaikki Skrollin tilaukset ovat määräaikaisia, joten myös vuosikertatilaajien on uusittava tilauksena. Kestotilauksiin ei meillä valitettavasti vielä ole varaa, mutta ehkä vuoden päästä jo on!

Skrolli

Tietokonekulttuurin erikoislehti

Yhteydenotot toimitus@skrolli.fi
ircnet — #skrolli

Päätoimittaja Ville-Matias Heikkilä
Toimituspäällikkö Toni Kuokkanen
Toimitussihteeri Ninnu Koskenalho
Taiteellinen johtaja Risto Mäki-Petäys
Mediamyynti Jari Jaanto
Talous Anssi Kolehmainen

Muu toimitus Lauri Alanko, Mitol Berschewsky, Tapio Berschewsky, Mikko Heinonen, Jukka O. Kauppinen, Ronja Koistinen, Sade Kondelin, Teemu Likonen, Kalle Viiri

Tämän numeron avustajat Filip De Haas (Otium), Ville-Veikko Heinonen, Juha Impivaara, Panu Kalliokoski, Jari Komppa, Henrik Kärkkäinen, Antti Laaksonen, Tapio Lehtimäki, Pauli Marttinen, Jarto Nieminen, Jarkko Nääs, Annika Piiroinen, Visa-Valtteri Pimiä, Manu Pärssinen, Ville Ranki, Mikko Rasa, Henna Ruohonen, Teija Tuhkio

Julkaisija Alternative Party ry

Painopaikka Tammerprint, Tampere,
ISSN 2323-8992 (painettu)
ISSN 2323-900X (verkkojulkaisu)

Kannen kuva:
Filip De Haas (Otium)

YMPÄRISTÖMERKKI
MILJÖMÄRKT
441 878
Painotuote

Skrolli Jenkkilässä

Skrolli pääsi elokuussa mukaan Aalto Entrepreneurship Societyn järjestämälle matkalle Piilaaksoon. Reissun tarkoitus oli tutustuttaa yrittäjähenkisiä opiskelijoita paikalliseen yrityskulttuuriin.

Teksti: Jari Jaanto Kuvat: Jari Jaanto, Tapio Lehtimäki, Jyri Rasinmäki

Yhteensä 20 opiskelijaa teekkareista kauppatieteilijöihin vieraili matkan aikana noin 20 yrityksessä, joista valtaosa oli teknologia- tai mobiilisovellusyrityksiä. Vierailukohteina oli myös pari yrityskiihdyttämöä, lakiyritys ja Stanfordin yliopisto.

Piilaaksossa tietotekniikka on työkalu ja mahdollisuus. Sen käyttäminen tuntuu olevan jopa arkipäiväisempää kuin Suomessa. Twitteriä ja mobiilipalveluita käytetään paljon, ja lähes jokaiselle päivittäiselle toiminnolle näyttää olevan oma mobiilisovelluksensa. Hashtagit näkyvät voimakkaasti mainoksissa ja katukuvissa. Tietokoneista suosituin näytti olevan MacBook Pro ja puhelimista iPhone. Myös Android-puhelimia näkyi paljon.

Parhaan kuvauksen Piilaaksosta heitti ilmoille talollamme vieraillut Mårten Mickos: ”Tämä Piilaakso on kuin aikuisten Otaniemi.”

Viis lomista, nyt koodataan!

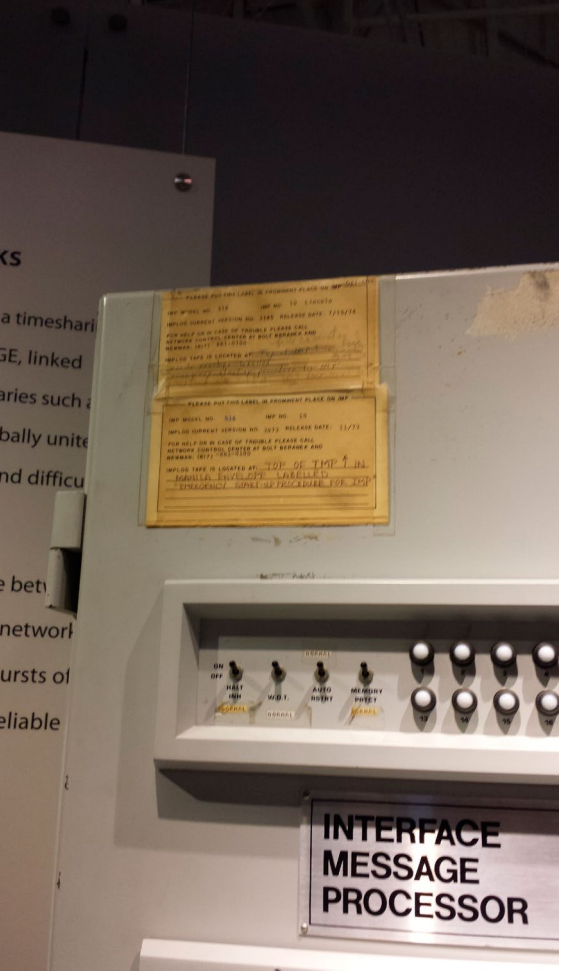
Piilaakson yrityskulttuuri hakee vertaistaan. Kahden viikon tutustumisen perusteella lähes jokaisen yrityksen perustajat ja toimitusjohtaja ovat koodereita. Esimerkiksi Kiva-järjestön perustaja Matt Flannery työskenteli aiemmin Tivolla Java-ohjelmoijana mutta lopetti työt, kun televisiolähetyksen tauottaminen ei tuntunut elämäntehtävältä. WhatsAppin perustajat koodasivat itse sovelluksensa ensimmäiset versiot, ja nykyinen toimitusjohtaja ja perustaja ohjelmoi yhden mobiilisovelluksista. Evernoten perustaja aloitti ohjelmointiuransa Atari-tietokoneella 1980-luvulla.

Listaa voi jatkaa lukemattomien muiden yritysten johtajilla, kuten Foursquaren, Twitterin, Facebookin ja Googlen. Jopa eräs tapaamani sijoittaja hehkutteli tekevänsä nykyään kaiken Scala-kielellä, kehuen sen maasta taivaisiin. Tämä on mukavaa vaihtelua, sillä meillä sijoittajat eivät ajanpuutteen takia yleensä koodaa. Piilaaksossa jokainen tuntui olevan kädet syvällä tekemisessä kiinni.

Kun perjantai-iltana kahdeksan aikaan olimme poistumassa eräästä lakitoimistosta, jatkoi moni vielä ahertamista työpisteillään. Ensisijainen syy ei ollut työmäärässä vaan siinä, että ihmiset ihan oikeasti rakastavat työtään. Sama näkyi monessa yrityksessä: ihmiset kertoivat tekevänsä juuri sitä, mitä haluavat.

Foursquarella tapaamamme ohjelmistoinsinööri kertoi, ettei ihmisiä niinkään johdeta vaan asiat keskustellaan aikuismaisesti halki. Hän kutsui Foursquarea aikuisimmaksi työpaikaksi, jossa oli työskennellyt. Facebookin ohjelmistokehittäjä puolestaan kertoi, että kaikki tekevät kovaa työtä ja yrityksen hyväksi tehty työ palkitaan – esimerkiksi onnistunut koodin optimointi, niin että se käyttää 2 % vähemmän prosessoritehoa. Piilaakson työkulttuurissa ihmiset tuntuvat olevan erityisen sitoutuneita yrityksensä tavoitteisiin ja haluavat tehdä parhaansa sen puolesta.

Työkulttuurissa näkyi hyvin myös henki ”hoidetaan hommat nyt saman



Interface Message Processor – pala alkuperäistä Arpanetia.

Piilaakson kulttuurin kiteyttävä NeXT Cube, jolla www kehitettiin.

tien pois alta”. Kokouksia ei sovita myöhemmäksi vaan asiat tehdään heti. Myös lomat tuntuivat olevan vieras käsite. Piilaakson suomalaiset kertoivat, että meikäläisiä pitkiä kesälomia ihmetellään rapakon takana. Monissa yrityksissä lomat saattavat peruuntua viime hetkellä esimerkiksi ison tilauksen myötä.

Autoilua Amerikan malliin

Piilaaksossa ollaan monessa asiassa Suomea edellä. Hyvä esimerkki on paikallisen taksijärjestelmän kanssa kilpaileva Uber. Uber yhdistää kuljettajat ja asiakkaat mobiilisovelluksen kautta. Kuka tahansa voi rekisteröityä Uber-kuskiksi. Kuljettajat saavat iPhonen ja laturin Uberilta, mutta auton vakuutuksista, bensasta ja muista kustannuksista he vastaavat itse. He voivat ajaa silloin, kun huvittaa.

Kun kuljettajat lähtevät ajoon, he käynnistävät matkapuhelinsovelluksen, joka välittää auton gps-koordinaatit Uberille. Asiakkaat voivat tilata auton luokseen tai valita kartalta jonkin lähikohteen. Sovellus kertoo, kuinka kauan lähimmältä kuljettajalta kestää saapua paikalle. Maksusta ei tarvitse huolehtia kyydistä noustessa, sillä matkan jälkeen Uber laskuttaa asiakasta automaattisesti. Kuitin saa sähköpostiin, ja mukana tulee kartta tehdystä matkasta.

Julkisessa terveydenhoidossa ja sosiaalipalveluissa taas ollaan selvästi Suomea jäljessä. San Franciscon kaduilla vaeltelee lukemattomia kodittomia, joilla ei ole mitään paikkaa minne mennä. Myös lääkärin saaminen voi olla vaikeaa. Eräs Uber-kuski totesikin, että Yhdysvalloissa selvitäkseen on työskenneltävä ahkerasti, sillä pohjoismaista sosiaaliturvaa ei ole. Suomalaiset ovat tällä alalla luomassa uusia palveluita: Ari Tullan ja Tapio Tolvasen perustama yhdysvaltalainen yritys BetterDoctor auttaa löytämään hyväksi todettuja lääkäreitä tiettyihin vaivoihin.

San Franciscossa voi liikkua linja-autolla, ja Piilaaksoon pääsee junalla, mutta alueen sisällä on helpointa liikkua Uber-taksilla. Pidempiä matkoja varten voi vuokrata auton lentokentältä. Steve Jobsin talolle ei kannata tehdä pyhiinvaellusta, sillä naapurit tulevat heti ajamaan pois.

Alueella järjestetään päivittäin useita tapaamisia ja kaikkea on tarjolla tietokonegrafiikasta palvelinten säätämiseen. Suosittelen käymään niissä ja tutustumaan mahdollisimman moniin ihmisiin. Tapaamisista saa tietoa meetup.com-sivulta.

Hackathlonin tunnelmia

Matkamme aikana järjestettiin TechCrunch Disrupt -tapahtuma, jossa teknologia-startupit esittelivät tuotteitaan. Tapahtumaan kuului 24 tunnin hackathlon, jossa eri ohjelmoijaryhmät kehittelivät tuotteitaan.

Päädyin tapahtumaan sattumusten kautta. Olimme viettämässä sunnuntai-iltaa, kun porukkaamme kuulunut Albert juoksi paikalle ja kertoi mahdollisuudesta lähteä TechCrunch Disrupt Hackathlon -tapahtumaan koodaamaan. Hetken emmittyäni tartuin mahdollisuuteen, ja lähdimme kohti tapahtumapaikkaa. Tietokonetta minulla ei ollut mukana, mutta Albert tarjoutui hakemaan sen asunnol-

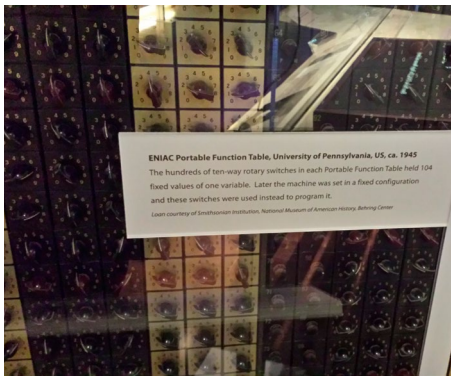
Skrolli ja jutun kirjoittaja Golden Gatella.

tamme.

Kiertelin tapahtumasalissa ja keskustelin osallistujien kanssa ohjelmoinnista. Puhuimme hyvin samaa kieltä. Eräälle henkilölle piirsin pikaisesti arkkitehtuurin hänen ideoimaansa ”blogilinkkien

Piilaakson trendejä

- Autojen, tavaroiden ja palveluiden yhteisöllinen jakaminen
- Fyysisten laitteiden, etenkin autojen, kytkeminen nettiin
- Hyvinvointiteknologia
- Älykodit
- Maaseudulle ja kehittymättömille alueille tuotava teknologia
- Big Data
- Maksujärjestelmät



Eniac, ensimmäinen yleiskäyttöinen elektroninen tietokone.


3D-mallintamisesta tuttu Utah Teapot.

klikkausprosenttimittariin", minkä jälkeen jatkoin matkaa salissa. Osa porukasta oli enemmän, osa vähemmän kokenutta. Kaikkialla aherrettiin erilaisten ohjelmien parissa, ja niitä riitti yksinkertaisista web-sovelluksista monimutkaisiin graafisiin sovelluksiin.

Henkilökemioiden perusteella päädyin lopulta ryhmään, jossa oli alun perin kaksi henkilöä. Ryhmä teki startupbets.com-nimistä palvelua, jossa voi ostaa virtuaalirahalla startup-yritysten osakkeita ja siten veikata, mitkä niistä tulevat menestymään ja mitkä eivät. Projekti kuulosti olevan sopiva yhden yön aikana toteutettavaksi.

Minusta tuli ryhmän web-ohjelmoija. Piirsimme nopeasti käyttöliittymän, jota ryhdyin toteuttamaan, samalla kun toiset rakensivat Pythonilla taustajärjestelmää ja parin palvelun rajapintakoodia. Illan aikana totesin, että tarvitsisimme myös graafikon, koska lyhyessä ajassa emme ehtisi muuta kuin koodaamaan. Huutelin WhatsApp-kanavallamme, ja kutsuun vastannut graafikko saapui kolmessa tunnissa paikalle lainakone kainalossaan. Yön aikana hän piirsi grafiikat ohjelmaamme ja minä koodasin.

Yöruokana oli tietenkin pizzaa, ja tarjoilupöytien ympärille muodostui nopeasti jonoja. Ryhmämme Amerikan-pojat kävivät nukkumassa pari tuntia, mutta minä ja graafikkomme Jyri jatkoimme sisulla läpi yön. Herättyään paikalliset ihmettelivät yöllisiä aikaansaannoksiamme. Softa saatiin jotenkuten valmiiksi määräaikaan mennessä. Lopuksi jokainen ryhmä esitteli ja mainosti omaa tuotostaan minuutin ajan. Tuotoksia oli yhteensä lähes 800, ja niiden esittelyyn kului aikaa noin kymmenen tuntia.

Tapahtuma jatkui kaksipäiväisenä startup-esittelytilaisuutena. Monet yritykset esittelivät myös rajapintojaan, joille pääsi ohjelmoimaan. Merkittävin näistä oli Chevroletin paikalle tuoma auto, jolle hackathlonissa sai koodata. Yksi Piilaakson trendeistä olikin sovellusten kehittäminen autoille. Esimerkkiautossa oli kaksi näyttöä, joista toinen sijaitsi keskikonsolissa ja toinen mittareiden keskellä. Myös auton keräämää dataa oli paljon, ja sitä analysoimalla pystyi optimoimaan auton suorituskykyä.

Historian koneiden hurinaa

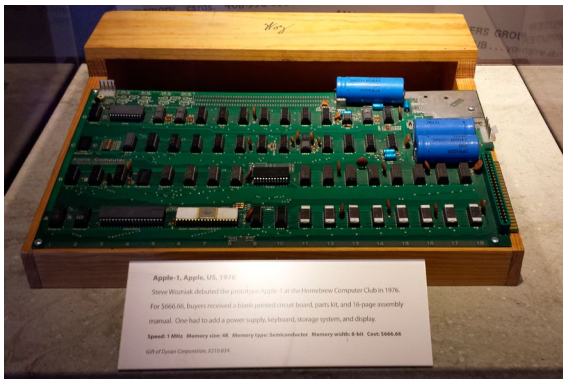
Vierailin Mountain View'ssä sijaitsevassa tietokonehistorian museossa (Computer History Museum). Käynti oli todella hieno kokemus, sillä edustettuna oli koko tietotekniikan historia aina helmitauluista nykykoneisiin.

Itselleni tärisyttävin kokemus oli päästä näkemään Eniacista jäljellä oleva osa. Siinä näkyi erinomaisesti, kuinka 1940-luvulla ratkottiin tietoteknisiä ongelmia. Putkia ja säätimiä oli reippain mitoin, mutta koneen arkkitehtuuri vaikutti melko yksinkertaiselta. Osat olivat tietysti suurikokoisia nykyisiin verrattuna. Yhden moduulin kyljessä oli valtava määrä säätövipuja, joilla funktiotaulut säädettiin kuntoon.

Museosta löytyi myös Xerox Alto, Next Cube, Apple-1, Honeywell Kitchen Computer, VAX 11/780, IBM 360, PDP-1 ja monta muuta, joita on aiemmin tullut ihasteltua vain kuvien kautta. Myös 1980-luvun robotit olivat hyvin esillä. Internetin historiaa edusti paikalla ollut Interface Message Processor (IMP), joka on osa alkuperäistä vuoden 1969 ARPANET-verkkoa.

Museon lisäksi paikalta löytyi Nasan tukikohta. Siellä oli näytillä Mercury-avaruussukkula sekä avaruusasema, jonka sisällä pääsi käymään. Osa avaruusaseman osista oli todella käynyt avaruudessa, tai niin ainakin kyltit väittivät. Vierailuun kannattaa varata vähintään kolme tuntia aikaa. Tutustumisen arvoinen paikka on myös Nasa Ames Research Center, joka sijaitsee Mountain View'n ja Sunnyvalen välissä.

Piilaakso ja San Francisco ovat hienoja paikkoja, mutta jos on kiinnostunut startup-yrityksen perustamisesta, ei oppeja tarvitse lähteä sinne saakka hakemaan. Piilaaksossa kannattaa käydä tutustumassa tietotekniikan kehitykseen ja mahtavaan teknologiakulttuuriin.



Apple I

Alkuperäinen Pong-kolikkopeli.

Kohteena olleet yritykset ja yhteisöt

- 500 Startups
- Amprius
- Bootstrap Labs
- DLA Piper
- Detecon
- Dropbox
- Electronic Arts
- Eucalyptus Systems
- Facebook
- Foursquare
- Google
- IDEO
- Idean
- Kiva
- Navis
- Nexit Ventures
- Palantir
- Sourcebits
- Stanford University
- Uber
- WhatsApp
- Yiftee

Rakenteluekstra

Henrik Kärkkäinen

Tekisinkö itse tietokonelehden?

Puhukaamme tietotekniikkalehden tekemisestä, tuosta salatieteistä vaarallisimmista, ja tiestä, jonka harva on viime aikoina kulkenut joutumatta naurunalaiseksi. On aika paljastaa täydellisen tietokonelehden tekemisen resepti.

Tietokonelehden tekeminen on nykyisin vähän vaikeaa. No joo, ei valehdella heti alussa. Paljon vaikeaa. Kaksi asiaa on nimittäin vähän toisin kuin kolmekymmentä tai kymmenenkin vuotta sitten. Tai no, ei valehdella vieläkään. Paljon toisin. Ne asiat ovat tietokone ja lehti.

Nykyisin kutsutaan tietokoneiksi hopeanvärisiä suorakaiteita, joissa on hedelmän kuva. Harvalla on silti kykyä tai halua avata sen koteloa.

Aiemmin lehti tarkoitti prosessoitua kuollutta puuta. Nyt se tarkoittaa myös klikkaaseksihelle-nettisivuja sekä lukijoiltaan piilotettuja tablettitiedostoja, joiden myyntivoitosta kolmannes menee eräälle hedelmätukulle tai sen liikemallia peesaileville firmoille.

Maailma muuttui. Yhä useammalla on tietokone, mutta samalla niistä ymmärretään yhä vähemmän. Tee siinä sitten tietokonelehteä.

Kaksi reittiä

Yritetään silti. Tietokonelehteä voi tehdä kahdella tavalla, joko vetoamalla laajaan ipadonparastaikinä ja tekniikantarkoitusonhelpottaaelämää -ryhmään tai matriisitulostimella tehtyjä ohjelmistolistauksia palvovaan, sisäänpäin lämpiävään kansanosaan.

Ensimmäisen resepti on nopeasti käsitelty: Paljon hedelmäsisältöä, isoja kuvia, kallista paperia ja näyttävä esillepano. Sitten toivotaan parasta.

Sitten on se kivisempi tie. Esitellään sen tärkein ja salaisin ainesosa ensimmäisenä. Se on kohderyhmä eli ne, jotka keitoksesta maksavat.

Kohderyhmä löydetään sisällöllä. Siispä tarvitaan lukujuttuja eli tarinoita bittimaailman hämmästyttävistä syövereistä. Kuten tarina Ron Austinin ja Kevin Poulsenin kiinni jäämisestä 80-luvulla. Esimerkiksi sellainen, jonka lukijat tuomitsivat ei-toivotuksi sisällöksi erään tietokonelehden alkutaipaleella vuonna 1984.

> *Lukijoiden ohella tärkein osa on goodwill.*

Oikean maun perässä

Seuraavaksi lehteen tulee rakentelua. Kannattaa olla varovainen, sillä sen voi toteuttaa väärin.

Prosessori + tuulettimet + tahnat + emolevy + muistit + näytönohjain + massamuisti + kotelo + erikoisuudentavoittelu-pci-e-kortit = pc. Tämä on väärin, sillä resepti on liian mainstream.

Raspberry Pi + Pong-koodin asennus = hack. Oikein, sillä virittely. Näettehän eron?

Tarvitaan myös huumoria, mielellään sisäpiirisellaista. Huonolaatuiselle tuumorille (sic) voi vaikka perustaa oman palstansa, ettei se saastuta koko julkaisua. Esimerkki: Tetris on klassinen retropeli. Eli retris.

Nii. Ei tuollaisia kehtaa ympäri lehteä ripotella.

Myös koodaamista on syytä opettaa. Nykyisin se on hankalampaa kuin ennen, sillä juuri nyt ei ole tarjolla yhtä paljon ilmaisia, tulevien it-pomojen tarjoamia ohjelmistolistauksia. Lisäksi ohjelmien keskipituuden muutos johtaisi painoteknisiin ongelmiin, kun fonttikoko lähentelee prosessorien viivaleveyksiä.

Sarjakuva on bonusta. Se saa olla mielellään yksiruutuinen, ja sen voi lehden suosion kasvaessa venyttää kolmiruutuiseksi. Ja sitten kutistaa taas yhteen ruutuun ja saada haukut juuriensa unohtamisesta.

Puuta vai bittiä

Lukijoiden ohella tärkein osa on goodwill. Sitä vähentävät lehdentekijöiden oleminen ison korporaation palkkalistoilla, klikkausotsikot sekä aiemmin mainittu runsas omppusisältö. Goodwilliä taas lisäävät sisällön askartelu yöaikaan ja lomilla vapaaehtoisvoimin.

Viimeisenä sisältöosana tulee jakelukanava. Digitaalista asiaa käsittelevän paketin voi tarjoilla mahdollisimman analogisesti eli puita kaataen, tai vaihtoehtoisesti Yhdysvaltain kansallisen turvallisuusviraston laajennetun intranetin välityksellä.

Näistä kumpikaan ei ole väärin, mutta ensimmäinen on enemmän oikein, koska paperi on tablettia kevyempi kannateltava pöntöllä istuttaessa. Jossain vaiheessa tietotekniikkalehden kehitystä on nimittäin saavutettava vessapaperivirstanpylväs, tai sitten lehdentekijän on syytä alkaa katsella muita töitä.

Täysin asiaan liittymättä: Missä sinä muuten luet tätä nyt?

Lukijan tunneside viimeistellään antamalla hänelle ulkoinen vihollinen. Sellaisen järjestäminen on hankalaa, mutta joskus vihollinen järjestää itse itsensä. Esimerkiksi tekemällä lehtiuudistuksen.

No, olisiko tällaisen lehden tekeminen mahdollista? Ei ja kyllä.

Ei siksi, että retrolehdenkin on oltava moderni. Uutta 20–30 vuoden takaisessa maailmassa olevaa lehteä kukaan ei halua lukea. Ne tehtiin jo. Kyllä siksi, että jotkut pääsevät aika lähelle.

Kirjoittaja on kirjoittanut useampiin Sanoman lehtiin – mukaan luettuina monien mielestä kehnohkosti uudistunut sellainen ja klikkaaseksihelle-otsikoiva sellainen.

Ohjelmistokäännöksen kivikot

Tietotekniikan käyttäjä huomaa yhäkin kielikukkasia suomenkielisissä ohjelmissa. Selvitimme, mistä käännösongelmat saavat alkunsa ja miten ohjelmistoja ylipäänsä käännetään.

Teksti: Mikko Heinonen Kuvat: Manu Pärssinen

”Valitse Asetukset, napsauta sitten painiketta Tiedot ja valitse Ota käyttöön”, toteaa ohjelmauutuuden käyttöohjetiedosto. Käyttäjä sen sijaan näkee edessään ihan erilaisen ikkunan, jossa näkyy ”Valinnat”, ”Lisätiedot” ja ”Käytä”. Käyttöohjeesta apua etsivä käyttäjä kiroaa mielessään, eikö se kääntäjä ole edes nähnyt tätä ohjelmaa?

Lyhyt vastaus on, että ei useinkaan ole. Ja vaikka olisikin, hänellä ei ehkä ole ollut lupaa muuttaa virheellistä käännöstä. Jotta voisimme ymmärtää, miksi tällaista tapahtuu, pitää aloittaa varsin kaukaa.

Kielenkääntäjästä tulee useille yhä mieleen kammiossaan istuva ammattilainen, joka ansaitsee elantonsa kääntämällä kaunokirjallisia teoksia. Televisio-ohjelmia ja elokuvia taas suomentavat AV-kääntäjät, jotka nousivat julkisuuteen suurten mediatalojen ulkoistettua käännöstoimintansa. Kolmas ammattiryhmä käännösalalla ovat tekniset kääntäjät, joiden vastuualueita ovat käyttöohjeet, viralliset asiakirjat ja ohjelmistojen lokalisoinnit. Jotkut moniottelijat tekevät kaikentyyppistä käännöstyötä, mutta useimmat kääntäjät keskittyvät vain yhteen lajityyppiin.

Kone muistaa, ihmistä tarvitaan

Tekninen kääntäminen eroaa kirjallisuuden kääntämisestä ja AV-kääntämisestä siinä, että se on hyötynyt eniten tekniikan kehityksestä. Koska esimerkiksi käyttöohjeet ja ohjelmistot sisältävät runsaasti samoja ilmauksia ja käytäntöjä, on niin sanottujen käännösmuistien käyttö muodostunut keskeiseksi työkaluksi teknisessä kääntämisessä. Käännösmuisti tallentaa kaikki tiettyä projektia koskevat semanttiset yksiköt – esimerkiksi yksittäiset sanat, lauseet tai virkkeet – suureen tietokantaan, josta niitä voidaan myöhemmin hakea.

Ihannetilanteessa käännösmuisti on erinomainen työkalu. Sen avulla samaa projektia voi tehdä vaikka usea kääntäjä samanaikaisesti ja tyyli pysyy yhtenäisenä, kunhan kaikki noudattavat vanhojen käännösten sanastoa ja ilmaisutapaa. Myöhempien päivitysten teko helpottuu huomattavasti, kun kaikki aiemmat käännökset ovat käytettävissä.

Käännösmuistin käyttöä ei ole syytä sekoittaa konekääntämiseen, vaikka niillä samoja piirteitä onkin. Käännösmuisti ei osaa kääntää automaattisesti kuin täsmälleen samanlaisina toistuvat jaksot. Kaikkeen muuhun se osaa tehdä vain ehdotuksia, joita ihmisen pitää muokata. Englanninkielinen termi *Computer Assisted Translation* tarkoittaa juuri tätä. Suomen kaltaisen kielen tapauksessa erona on myös se, että tietokoneavusteinen käännös nopeuttaa kääntäjän työtä, siinä missä konekäännöksen sijamuotojen ja sanavalintojen korjailu lähinnä hidastaa sitä.

Rengistä isännäksi

Kääntäminen maksaa rahaa, joten yrityksen taskulaskinosasto haluaa säästöjen nimissä käyttää samoja käännöksiä aina kun mahdollista. Tämä on myös järkevää kielen yhtenäisyyden kannalta ja helpottaa uusien kääntäjien ottamista mukaan työhön.

Jossain vaiheessa käy kuitenkin niin, että käännösmuistista tulee lopputulosta rasittava riippakivi. Osa suurten ohjelmistojen käännöksistä saattaa olla yli 15 vuotta vanhoja. Tietotekniikan sanasto on tällä välillä voinut muuttua moneen otteeseen, ja toisaalta ei ole lainkaan varmaa, että kaikki alkuperäistä ohjelmaa kääntäneet ovat tienneet, mistä tarkalleen on kysymys.

Vaikka nykyinen kääntäjä olisi perustellusti eri mieltä käännösmuistin kanssa, vanhaa käännöstä on usein pakko käyttää yhteneväisyyden nimissä – tai ainakin siksi, ettei sen muuttamisesta makseta kenellekään. Monet yritykset vain ajavat vanhat käännökset sisään automaattisesti eivätkä tarkista, pitävätkö ne vielä paikkansa. Niinpä vuoden 2013 ohjelmistokin voi ihan sujuvasti jutella käyttäjälle levykkeistä tallennusvälineinä.

Tarpeeksi paisuttuaan käännösmuisti alkaa sisältää samalle asialle erilaisia käännöksiä. On jopa voitu toteuttaa projekti, jossa ikivanhoja käännöksiä on korjailtu nykyaikaisemmiksi mutta vanhat on jätetty muistiin kummittelemaan. Aloitteleva kääntäjä, joka ei tunne käännettävää ohjelmistoa, ei luonnollisestikaan tiedä, mikä käännösmuistin tarjoamista vastineista pitäisi valita. Usein aloittelevan kääntäjän on myös palkannut yritys, joka on tarjonnut työstä halvimman hinnan.

Lue se manuaali, jos pystyt

Riippuvuussuhteet tulevat vieläkin monimutkaisemmiksi, kun samaan projektiin kuuluvat sekä ohjelmisto että käyttöohje. Ohjelmisto luonnollisesti käännetään ensin, minkä jälkeen käyttöohje usein kilpailutetaan erikseen ja annetaan jollekin toiselle ryhmälle tehtäväksi.

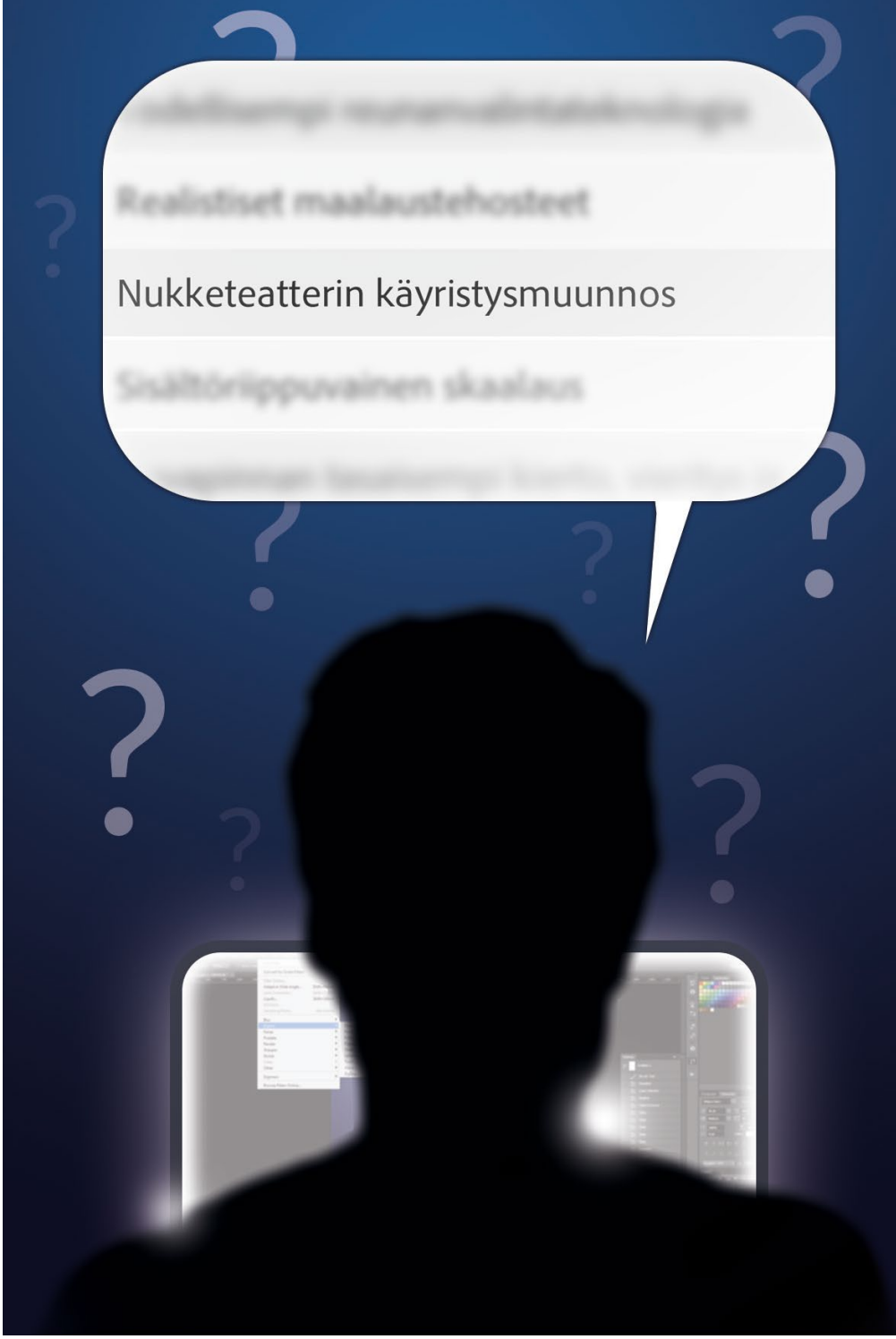
Käyttöohjeen kääntäjille annetaan tavallisesti vain lista, jossa on lueteltu ohjelman käyttöliittymän käännökset. Vain ani harvoin heille annetaan itse ohjelmaa. Tätä listaa on sitten noudatettava kuin pyhää kirjaa, ja myös sen virheet on jäljennettävä. Aina ei ole tietoa käyttöliittymästä, jolloin kääntäjä arvailee parhaansa mukaan ja syntyy tämän artikkelin aloituskappaleen mukaisia tilanteita.

Joskus kääntäjätkin ylisuorittavat ja huomauttavat ohjelmistossa olevista virheellisistä teksteistä. Yleensä se ei johda mihinkään, sillä käyttöohjetta käännettäessä ohjelmisto on jo ”kiinni”, eikä siihen enää tehdä muutoksia. Parhaassa tapauksessa muutos saattaa tulla seuraavaan versioon, mutta kieliasiat ovat harvoin niin keskeisiä, että asiaan jaksettaisiin panostaa.

Seuraava mahdollisuus virheellisen käännöksen korjaamisen tulee vastaan silloin, kun jokin tekninen seikka pakottaa kirjoittamaan kyseisen kohdan uusiksi. Tällöin virheellinen käännös onkin jo usein levinnyt niin moneen paikkaan, ettei sen täydellinen poistaminen ole enää mahdollista.

Termistöjen sekamelskaa

Yrityksen viestintä on tärkeä osa sen julkisuuskuvaa, ja moni on ottanut kielen mukaan imagonsa luomiseen. Joskus tuntuu, että markkinointipuolella laaditaan kilpaa erilaisia tyylioppaita ja kehitetään omaa sanastoa. Sen seurauksena samoja asioita käännetään monella eri tavalla. Esimerkiksi icon voi olla suomeksi kuvake, ikoni tai symboli. File on tiedosto tai arkisto, riippuen siitä, missä käyttöjärjestelmässä ollaan. Sama koskee monia muitakin keskeisiä termejä.



Kääntäjän näkökulmasta tämä aiheuttaa runsaasti ongelmia, sillä usein esimerkiksi ohjelmiston Windows- ja Mac OS -versiot tulevat käännettäviksi samaan aikaan. Alkukielinen teksti ei erottele eri käyttöjärjestelmiä, vaikka käännöksissä niin tehtäisiin. Tällöin kääntäjän on ratkaistava, mitä varianttia käytetään, ja usein markkinaosuudet ratkaisevat.

Termistöjen suhteen parhaan arvosanan ansaitsee Microsoft. Ei siksi, että sen käännökset välttämättä olisivat mitenkään ylivertaisia. Syynä on se, että Microsoft on asettanut lähes kaikki tekemänsä käännökset vapaasti saataville. Microsoft Language Portal -sivustosta löytyvät kaikkien tärkeimpien ohjelmistojen käännökset kaikille kielille. Sivuston hakutoiminnon tuloksista on mahdollista tutkia, missä tuotteessa kutakin käännöstä on viimeksi käytetty. Näin on helppoa poimia omaan käyttöön ajantasainen viittaus esimerkiksi Windows 8:n käyttöliittymään.

Rakas, rikas äidinkielemme

Aivan oman mausteensa käännöskeitokseen antavat suomen kieli ja sen taivutusjärjestelmä. Jokainen suomenkielisiä ohjelmia käyttänyt on varmasti törmännyt sanoihin ”kohde” ja ”järjestelmä”, jotka eivät yleensä tarkoita yhtään mitään tilanteissa, joissa niitä käytetään. Kyse onkin tilkesanoista, joilla suomen taivutus saadaan sopimaan anglosaksiseen ympäristöön.

Tilkesanoihin päädytään useimmiten silloin, kun tekstissä on tavaramerkkejä tai muuttujia, jotka estävät normaalin taivuttamisen. Monet suuryritykset valvovat tarkasti tavaramerkkejään ja kieltävät ehdottomasti niiden taivuttamisen, koska joku ilmeisesti saattaisi tulkita, ettei taivutettu merkki enää olekaan rekisteröity.

Toisilla yrityksillä taas on tapana tehdä monia versioita ohjelmistostaan, jolloin vaikkapa ”SuperSystem 8000 Ultra-Professional” ja ”SuperSystem 7500 Piss-Poor” näkyvät kääntäjälle havainnollisena muuttujana ”%s” tai ”%ProdName”. Näin syntyy kukkasia kuten ”SuperSystem-järjestelmään” eli ’superjärjestelmäjärjestelmään’. Voin vakuuttaa, että tämä särähtää kääntäjänkin korvaan, mutta mitään muuta vain ei ole tehtävissä.

Sana ”kohde” taas pitää tulkita niin, että kääntäjä on saanut eteensä pelkän luettelon viittauksia, joiden tilalle ohjelmistossa voidaan vaihtaa yksi tai useampia sanoja. Tyyppiesimerkki tällaisesta on vaikkapa ”copy %1 from %2 to %3”. Tässä %1 on ilmeisesti tiedostonimi, joka ei kaipaa taivutusta, mutta ”%2 to %3” vaatisi jo suomessa sijamuotoja. On vain mahdotonta tietää, mihin kirjaimeen tai välimerkkiin ne loppuvat, joten tyyppi %2sta %3een ei onnistu. Puhumattakaan siitä, ettei muuttujan nimien perään useinkaan voi kirjoittaa suoraan ja että muuttujan kohdalle voi lauseessa tulla yksi tai useampia sanoja, mikä sotkee suomen yhdysmerkkisäännöt. Näin siis %1 kopioidaan kohteesta %2 kohteeseen %3 – taas kerran, varmuuden vuoksi.

Hyvin, huonosti ja hyvin huonosti

Ohjelmistokääntämisen ala on siis haasteita pullollaan. Toimittuani kokopäivätoimisena kääntäjänä kymmenisen vuotta olen ehtinyt istua vähän joka puolella pöytää: aloittelijana ihmettelemässä uutta outoa tuotetta, idealistisena maailmanparantajana tekemässä kehitysehdotuksia ja kyynisenä palkkaorjana lätkimässä pieniä katkelmia jo menetettynä tapauksena pitämäni projektin sekaan. Välillä olen ihmetellyt, kuinka ylipäätään mitään saadaan käännettyä tolkullisesti. Onneksi olen myös saanut ilahtua siitä, miten toisissa yhtiöissä kieliasioihin suhtaudutaan riittävällä vakavuudella.

Kokemukseni pohjalta olen laatinut aloitteleville ohjelmistoyrityksille oheisen taulukon, jota noudattamalla on helppoa munata ensimmäinen ohjelmistokäännösprojekti kokonaisvaltaisesti. Kaikki esimerkit ovat tosielämästä.

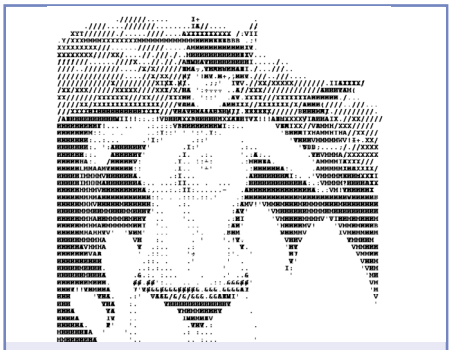
1. Tarjouskilpailu	Hyväksy halvin käännöstarjous. Älä suotta tarkista, onko tarjoajan suomenkielisten projektien hallinta esimerkiksi Kiinassa tai puhuuko kukaan sen henkilöstöstä kohdekieltä. Näin varmistat niin pitkän alihankintaketjun, ettei sen lopussa nälkäpalkalla nyhertävää kääntäjää kiinnosta kurja softasi vähääkään. Älä missään nimessä palkkaa alan koulutuksen saanutta ihmistä vastaamaan kieliasioista tai osta sellaisen palveluita. Ammattilainen saattaisi puuttua ohjelmistokehittäjien synnynnäisesti nerokkaisiin ratkaisuihin, jotka automaattisesti toimivat joka kielellä.
2. Käännettävä aineisto	Lähetä käännettäväksi mahdollisimman varhainen versio ohjelmistosta ja muuta sitä useita kertoja matkan varrella. Toimita kaikki lisäykset eri tiedostomuodoissa ja vaihda esimerkiksi muuttujien merkintätapaa joka kerta. Älä toimita mitään viiteaineistoa, josta kääntäjät voisivat tutustua aiempiin käännöksiin tai selvittää, mihin ohjelmistosi liittyy. Koska maksat heille, on heidän tehtävänsä selvittää tällaiset asiat parhaaksi katsomallaan tavalla eikä vain kopioida toisten jo tekemää työtä. Kieltähän kuitenkin käännetään kirjoittamalla samat sanat peräkkäin eri kielellä. Muista, että kääntäjät toimivat sitä paremmin, mitä tiukempi aikataulu heille annetaan. Lähetä käännettävää myös jouluaatoksi, jolloin itse vietät aikaa perheen parissa. Tämä korostaa juuri sinun hankkeesi tärkeyttä. Jos haluat todella varmistaa hyvän lopputuloksen, kehitä oma käännöstyökalu ja pyydä kääntäjiä syöttämään käännökset sinne. Tekniseen kääntämiseen tarkoitettuja ohjelmistoja on kehitetty vasta muutama kymmenen vuotta, joten yrityksesi kesäharjoittelija tekee varmasti paremman ensimmäisenä PHP-harjoitustyönään.
3. Kääntäjien kysymykset	Ohjelmistokehitys on sen verran kiireistä puuhaa, että humanistit ovat siinä vain tiellä. Jos teknisillä kääntäjillä on jotain kysyttävää tuotteistasi, kannattaa laittaa taloon juuri tullut markkinointisihteeri vastaamaan kysymyksiin. Missään nimessä ei kannata haaskata tuotteen tehneiden ohjelmoijien aikaa tällaiseen joutavuuteen, ja paras onkin jättää vastaamatta kääntäjien kysymyksiin.
4. Testaus	Ohjelmiston testaus ja laadunvarmistus kannattaa myös antaa halvimman tarjouksen tehneelle, jos nyt laatua edes halutaan varmistaa. Esimerkiksi Kaukoidässä toimiva testauskeskus pystyy tarjoamaan aivan ainutlaatuista osaamista suomalais-ugrilaisissa kielissä. Käännöksen laatu on helposti määritettävissä seuraavalla kaavalla: käännetyn tekstin pitää olla erilainen kuin alkuperäinen teksti mutta ei liian erilainen. Samoin kääntäjän työmotivaatiolle tekee hyvää, kun pääsee selittämään testaajalle neljännen kerran saman projektin aikana, mitä se sijamuoto taas tarkoittikaan. On myös tärkeää, että kaikki ohjelmiston merkkijonojen pituudet pakotetaan englannin kielen mukaisiksi. Minkään kielen sanat eivät voi olla pidempiä. Merkkirajoista ei kannata kertoa kääntäjille etukäteen, sillä he ehtivät kyllä lyhennellä käännöksensä myöhemmin, samaan hintaan tietysti. Tämä työllistää myös mukavasti testaajia, kun he saavat ottaa sadoittain ruutukuvia näyttöteksteistä, jotka ovat keskeltä katkenneita.

Merkistöt taiteen rakennuspalikoina

Kun jossain liikkuu tekstiä sähköisesti, liikkuu siellä usein myös merkkigrafiikkaa. Seuraa pikakatsaus muutamiin merkkigrafiikan muotoihin.

Teksti: Ville-Matias Heikkilä

Varhainen tietoliikenne oli yleensä tekstimuotoista, joten helpoin tapa esittää grafiikkaa oli muotoilla se tekstiksi. Merkkipohjaisuus on myös taloudellista, mistä syystä sitä suosittiin pitkään kaikenlaisissa tietokoneohjelmissa. Nykyään useimpiin paikkoihin voi ympätä millaisia kuvia hyvänsä, joten merkkigrafiikka on entistä useammin taiteellinen valinta teknisen välttämättömyyden sijaan.

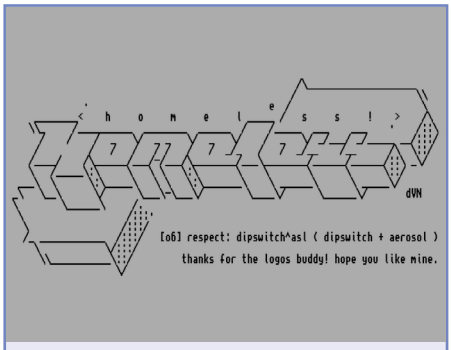


Kuvan tekijä tuntematon rtty.com

Baudot (1870)

Kaukokirjoitin keksittiin jo 1800-luvulla. Radioihin kytketyistä kaukokirjoittimista syntyi radioamatöörien käyttämä RTTY-yhteyskäytäntö eli Radio Teletype. RTTY:llä siirreltiin myös grafiikkaa.

Viisibittisiin baudot-koodistoihin kuuluvat vain isot kirjaimet ja numerot sekä joukko välimerkkejä. Merkkejä voi kuitenkin lyödä päällekkäin, mikä mahdollistaa tummemmat ja monipuolisemmat palikat.



Hiro Protagonist asciiarena.com

ASCII (1967)

128 merkkiä ja ohjauskoodia käsittävä ASCII on miltei kaikkien myöhempien merkistöjen perusta. Suurin osa tietoliikenteen merkkigrafiikasta onkin ollut nimenomaan ASCII-grafiikkaa. Erilaisia tyylejä on lukemattomia.

Amigalla syntyneessä ”oldskool-tyylissä” muodostetaan yhtenäisiä kuvioita keno-, kautta- ja muista viivoista. Kaikki fontit eivät kuitenkaan yhdistä viivoja toisiinsa aukottomasti.

Dan Farrimond teletextart.com

Teksti-TV (1976)

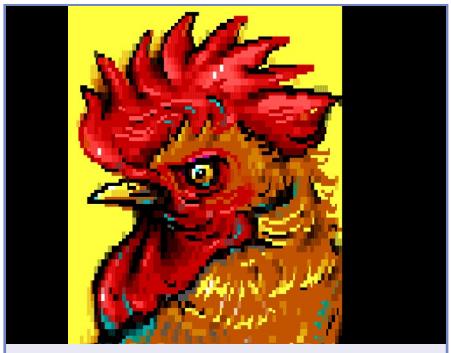
BBC:n kehittämä teksti-tv lähettää 40×25 solun kokoisia sivuja tv-kuvan lomassa. Solu voi sisältää joko kirjoitusmerkin, näkymättömän ohjauskoodin tai 2×3 pikselin grafiikkaelementin. Näin teksti-tv-sivuilla voi olla tekstin lisäksi myös karkeaa pikseligrafiikkaa.

Värejä on käytössä 8 sekä merkille että sen taustalle, mutta eriväriset alueet on erotettava toisistaan värinvaihtokoodeilla. Teksti-tv ei siis mahdollista ANSI-tyylistä väri-ilottelua.

Mermaid / Genesis Project

PETSCII (1977)

PETissä ja myöhemmissä kasibittisissä Commodoreissa käytettyyn PETSCIIhin kuuluu muun muassa kymmeniä erilaisia viiva- ja palkkielementtejä. Kaikki eivät tosin ole käytettävissä samaan aikaan pienten kirjainten kanssa. Merkkivärejä on C-64:llä tarjolla 16, ja ruudulle mahtuu 40 × 25 merkkiä.

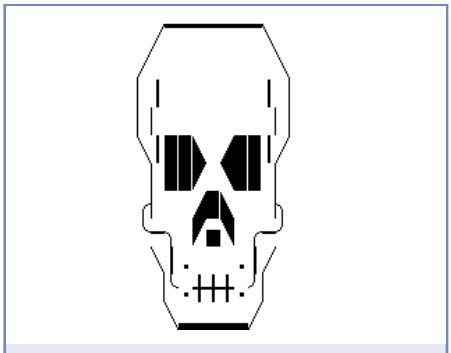


Ungenannt / Blocktronics sixteencolors.net

IBM PC (1981)

PC-merkkigrafiikkaa kutsutaan usein ANSI-taiteeksi sen käyttämien päätteenohjauskoodien vuoksi: kullekin merkille voi niillä valita päävärin 16:sta ja taustavärin 8:sta vaihtoehdosta. Tarjolla on vähemmän erilaisia palkkeja kuin PETSCIIssä, mutta sävytyselementeistä saa kauniita väriliukuja. Grafiikka on yleensä tarkoitettu 80 × 25 tai 80 × 50 merkin tekstitilalle, mutta teoksilla voi olla mittaa jopa kymmeniä ruudullisia.

Muutamat nykyiset ansieditorit ja useat pääte-emulaattorit tukevat laajempia väripaletteja, joita käytetään ainakin joidenkin MUDien grafiikoissa.



tw1tt3rart text-mode.tumblr.com

Unicode (1991)

Nykyisin kaikkialla käytettävä Unicode pyrkii yhdistämään kaikki merkistöt, mutta se ei vieläkään kata esimerkiksi PETSCIItä tai Videotexiä. Unicode tarjoaa kuitenkin joitakin uusia mahdollisuuksia, kuten tarkemerkkien kasaamisen päällekkäin. Mahdollisuudet kuitenkin vesittyvät siihen, että useimmat tasalevyisetkään nykyfontit eivät huomioi merkkigrafiikkayhteensopivuutta mitenkään. Lähinnä PC:stä tuttujen palikoiden voi olettaa toimivan edes jotenkuten samoin eri fonteissa. Useimmat merkkigrafiikkataiteilijat pitäytyvätkin mieluummin klassisissa merkistöissä.

Ohjelmointiparadigmat

Ohjelmointiparadigmat tarjoavat erilaisia näkökulmia ohjelmointiin. Ne säätelevät ohjelmoijan ajattelua ja vaikuttavat ohjelman lopputulokseen. Uusi paradigma on myös uudenlainen älyllinen haaste.

Teksti: Panu Kalliokoski Kuvat: Mitol Berschewsky, Panu Kalliokoski, Henna Ruohonen, Teija Tuhkio, Risto Mäki-Petäys

Ohjelmoijan työssä luodaan asioita tyhjästä. Ohjelmoijan tuottaman koodin alkuperä on aina ohjelmoijan ajatuksissa, vaikka ohjelmoitaessa toki myös muokataan vanhaa koodia tai kirjoitetaan uusi ohjelma esimerkkikoodin pohjalta. Ei ole harvinaista, että ohjelmoija tuottaa sivukaupalla ohjelmakoodia ohjeenaan vain oma intuitio ja kokemus sekä jonkinlainen määrittely siitä, miten ohjelman olisi tarkoitus toimia.

Koska ohjelmat syntyvät puhtaasti ajatuksista, ohjelmoijien ajattelutavoilla on valtava vaikutus siihen, millaisiksi ohjelmat tulevat. Ulkoisesti samalla tavoin toimivan ohjelman voi kirjoittaa äärettömän monella eri tavalla. Erilaiset lähestymistavat tuottavat eri pituisia ohjelmia ja ottavat huomioon erilaisia ohjelman laajennus- ja muutostapoja. Hyvinkin monimutkaiselta vaikuttava ongelma tai algoritmi saattaa toisella tavoin hahmotettuna olla niin yksinkertainen, että sen saa ratkaistuksi muutaman rivin ohjelmalla.

Ohjelmointiparadigma merkitsee yleistä ajatusmallia, jota pystyy soveltamaan mihin hyvänsä ohjelmointitehtävään.[1] Paradigmat on todistettu yhtäpitäviksi, niin että yhden paradigman ohjelmat on muunnettavissa automaattisesti toisen paradigman ohjelmiksi. Eri paradigmat toki tuottavat erilaisia ja eri tavoin siistejä ratkaisuja ongelmiin.

Ohjelmointiparadigmat kytkeytyvät ohjelmointikieliin siten, että kielen sanotaan ”edustavan” jotakin paradigmaa. Useimmat nykyaikaiset kielet ovat kuitenkin *moniparadigmaisia*, eli niillä on mahdollista kirjoittaa kohtuullisella vaivalla useamman paradigman mukaisesti. Tyypillinen valtavirtakieli (esimerkiksi C, Java tai Python) tukee siedettävästi ainakin funktionaalista, oliopohjaista ja proseduraalista lähestymistapaa. Paradigmat eivät aina sulje toisiaan pois, vaan ohjelman eri osat voivat olla eri tyyleillä kirjoitettuja.

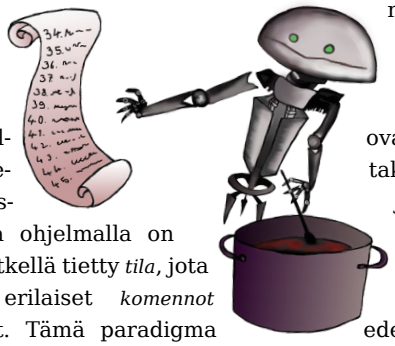
Erilaisten ohjelmointitapojen perusteellinen läpikäynti on valtava työ, minkä vuoksi tutustumme tässä vain muutamiin keskeisimpiin paradigmoihin. Tässä artikkelissa käsitellään seuraavia paradigmoja:

- Imperatiivinen paradigma, jonka alalajeja ovat strukturaalinen, proseduraalinen ja oliopohjainen ohjelmointi.
- Funktionaalinen paradigma, jonka alalajeja ovat perinteinen funktionaalinen, korkeamman kertaluvun funktioita hyödyntävä ja tietovuopohjainen ohjelmointi.
- Logiikkaparadigma, johon kuuluvat predikaattien monisuuntaisuus ja indeterminismi.

Harmillista kyllä, useimmat näistä paradigmoista eivät ole kovin hyvin määriteltyjä, eikä termien tarkoista merkityksistä vallitse yksimielisyyttä. Tätä moninaisuutta ei kuitenkaan ole mahdollista esitellä tässä jutussa, joten olen valinnut jokaiselle termille yhden melko laajasti hyväksytyn merkityksen.

Imperatiivinen paradigma

Imperatiivisessa paradigmassa ohjelman suoritus etenee selkeästi tietyssä järjestyksessä ja ohjelmalla on kullakin hetkellä tietty *tila*, jota ohjelman erilaiset *komennot* käsittelevät. Tämä paradigma on useille ohjelmoijille kaikkein tutuin, koska imperatiivista ohjelmointia tukevia (ja muut ohjelmointitavat hankalaksi tekeviä) kieliä on ollut olemassa jo hyvin pitkään. Imperatiivinen ohjelmointi soveltuu erityisen hyvin sellaisiin tehtäviin, joissa ohjataan suoraan laitteistoa, esimerkiksi ohjelmoidaan laiteajureita. Se sopii myös ohjelmiin, joiden täytyy tehdä asiat tietyssä järjestyksessä tai antaa takeita vasteajoista, muistinkulutuksesta tai muusta vastaavasta.



Imperatiivinen paradigma muistuttaa hyvin paljon todellisten tietokoneiden toimintaa. Monet varhaiset ohjelmointikielet toivat esiin käytössä olevan laitteiston ominaisuuksia. Ohjelmointikielen ja laitteiston läheinen yhteys teki imperatiivisesta ohjelmoinnista kaikkein luonnollisimman lähestymistavan.

Suuri osa moderneista ohjelmointikäytännöistä on kehittynyt erilaisista abstraktiokerroksista ja muista keinoista, joilla ohjelmoijat ovat pyrkineet järjestelemään ja selkeyttämään koodiaan sekä helpottamaan sen ylläpitoa. Ohjelmointikäytännöt ovat ohjeita, jotka rajoittavat sitä, millaisia ohjelmia saa kirjoittaa. Toisaalta ne tekevät ohjelmista lyhyempiä, helpommin ymmärrettäviä ja helpommin toimivaksi todistettavia.

```
Esistrukturaalinen (BASIC):  Strukturaalinen (FORTRAN):

10 INPUT a, b                READ(*,*)  a, b
20 c = a                     IF (a < b) THEN
30 IF a < b THEN 60             c = a
40 c = a MOD b                  a = b
50 IF c = 0 THEN 90             b = c
60 a = b                     END IF
70 b = c                     DO
80 GOTO 40                      c = MOD(a, b)
90 PRINT "syt: " b              IF (c == 0) EXIT
                                a = b
                                b = c
                             END DO
                             WRITE(*,*) 'syt: ', b
```

Listaus 1. Suurimman yhteisen tekijän etsintä. Ohjelman *tila* on molemmissa muuttujien a, b ja c sisältö kunakin suoritushetkenä sekä tieto siitä, millä rivillä ohjelmaa edetään. Toteutuskieli: BASIC, FORTRAN.

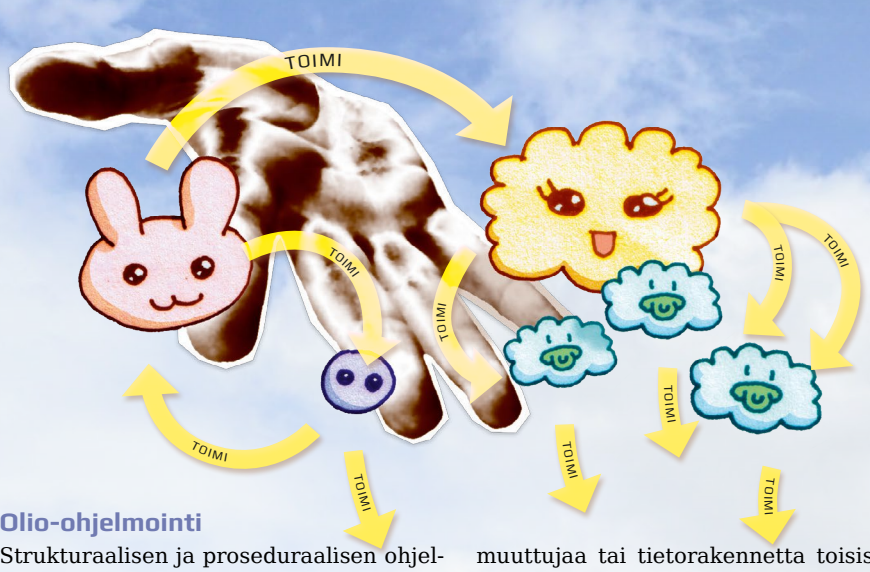
Strukturaalinen ja proseduraalinen ohjelmointi

Sekä strukturaalinen että proseduraalinen ohjelmointi ovat rajoituksia sille, milloin ohjelman suoritus saa hypätä uuteen kohtaan ohjelmakoodissa. Nykyisille ohjelmoijille strukturaalinen ja proseduraalinen lähestymistapa ovat lähes itsestään selviä. Niiden vastakohtaa, perinteistä GOTO- tai jmp-ohjelmointia, puolestaan ei kannattane mainita muuten kuin historiallisena kuriositeettina.

Strukturaalisessa ohjelmoinnissa edellytetään, että hyppyjä tehdään vain osana ehtorakenteita ja toistorakenteita, jotka eivät saa lomittua keskenään (listaus 1).

Proseduraalinen ohjelmointi puolestaan tarkoittaa, että ohjelma jaotellaan useiksi *aliohjelmiksi*. Jokainen näistä tekee yhden tarkkaan määritellyn tehtävän, ja aliohjelmat *kutsuvat* toisiaan tarvittaessa (listaus 2). Tällä tavoin koodia saadaan usein siistityksi, kun toistuva koodi saadaan siirretyksi omaan aliohjelmaansa. Mikä ehkä vieläkin tärkeämpää, ohjelmakoodiin saatiin proseduraalisen ohjelmoinnin myötä yleensä *nimi* erilaisille toiminnoille. Alkuaikoina harva kieli kuitenkin tuki sitä, että aliohjelma saisi kutsua itseään. Tämän vuoksi *rekursio* (tyypillinen funktionaaliseen paradigmaan kuuluva ohjelmointitekniikka) ei ollut mahdollinen.

[1] Sen sijaan ajatusmalleja kutsutaan yleensä *suunnittelumalleiksi (design patterns)*, jos ne soveltuvat vain tiettyyn tilanteeseen tai ongelmatyyppiin.



Olio-ohjelmointi

Strukturaalisen ja proseduraalisen ohjelmointitavan avulla pärjättiin muutamia vuosikymmeniä. Niiden vahvuutena on, että ohjelman suoritusjärjestys on helppo hahmottaa. Ajan mittaan kuitenkin huomattiin, ettei ohjelman suorituksen eteneminen ollut ainoa vaikeaselkoinen osuus ohjelmoinnissa. Oli tiettyjä asioita, jotka aiheuttivat säännöllisesti ohjelmointivirheitä suurissa ohjelmistoissa ja vaikeuttivat niiden ylläpitoa.

Imperatiiviset ohjelmat laskevat asioita päivittämällä ohjelman tilaa, käytännössä muuttujien arvoja ja tietorakenteiden sisältöjä. Huomattiin, että monet ohjelmavirheet johtuivat siitä, että ohjelman eri osat muuttelivat samaa muuttujaa tai tietorakennetta toisistaan tietämättä. Ajatellaan esimerkiksi aliohjelmaa, joka käy läpi lukulistaa (laskeakseen siitä jotain) ja tekee jonkin aliohjelmakutsun jokaisen listan elementin kohdalla: jos kyseinen aliohjelma sivuvaikutuksenaan muokkaa läpi käytävää listaa, lopputulos saattaa olla aivan muuta kuin ohjelmoija ajatteli.

Olio-ohjelmoinnissa tietorakenteita käsittelevät aliohjelmat yhdistetään tietorakenteisiinsa siten, että vain tietyt aliohjelmat, *metodit*, saavat muutella kutakin tietorakennetta. Tietorakenteesta tulee tällä tavoin itsenäinen yksikkö, *olio*. Kukin metodi pystyy osaltaan varmistamaan, että tietorakenne on eheässä ja toimivassa tilassa, kun ohjelman suoritus luovutetaan takaisin metodin ulkopuolelle. Samalla ohjelmien ylläpidettävyys paranee: tietorakenteen sisäistä toteutusta muutettaessa ei tarvitse korjata koko ohjelmaa käyttämään uusia rakenteita, vaan riittää, että päivittää rakenteen omat metodit.

Proseduraalisen ohjelmoinnin avulla tehtiin hyödyllisiä aliohjelmia, joita sitten käytettiin uudestaan ja uudestaan osana suurempia ohjelmointiprojekteja. Olio-ohjelmointi puolestaan avasi oven tietorakenteiden uudelleenkäytölle. Lisäksi olio-ohjelmoinnin myötä tuli huomattavasti helpommaksi kirjoittaa korkean tason koodia: oliota käsittelevä aliohjelma pystyy käyttämään sen metodeja sokeasti. Se toimii siten *minkä tahansa* olion kanssa, joka toteuttaa oikein samat metodit. Näin ohjelmien toiminnallisuutta saattoi jälkikäteen laajentaa uusilla tietotyypeillä (listaus 3).

Joskus luullaan, että oliopohjaista ohjelmointia varten kielessä pitää olla erityinen tuki erilaisille oliorakenteille. Todellisuudessa oliopohjaista koodia pystyy kirjoittamaan lähes kaikilla yleisessä käytössä olevilla ohjelmointikielillä (listaus 4) matalan tason konekielistä lähtien.

Olio-ohjelmointi soveltuu erityisen hyvin tilanteisiin, joissa halutaan tuottaa valmiskäyttöisiä tietorakenteita tai muita palveluita tulevien ohjelmien käyttöön. Oliopohjaista ohjelmaa pystyy helposti laajentamaan uusilla tietotyyppitoteutuksilla. Esimerkiksi peliin voi helposti lisätä uusia tekoälyjä tai esineitä.

```
Ilman aliohjelmia:

int a[] = {1, 3, 2, -7, 9, 4, -1, 5};
int length = sizeof a / sizeof a[0];
int step, i, j, tmp;
for (step = length; step /= 2;) {
  for (i = step; i < length; i++) {
    tmp = a[i];
    for (j = i;
       j >= step && tmp < a[j - step];
       j -= step)
      a[j] = a[j - step];
    a[j] = tmp;
  }
}


Aliohjelmien kanssa:

void shell_sort (int *a, int length)
{
  int step;
  for (step = length; step /= 2;)
    insertsort_by_step(step, a, length);
}

void insertsort_by_step(int step,
                        int *a, int length)
{
  int i;
  for (i = step; i < length; i++)
    insert_element_at(i, a, step);
}

void insert_element_at(int i, int *a, int s)
{
  int j, tmp;
  tmp = a[i];
  for (j = i; j >= s && tmp < a[j-s]; j -= s)
    a[j] = a[j - step];
  a[j] = tmp;
}
```

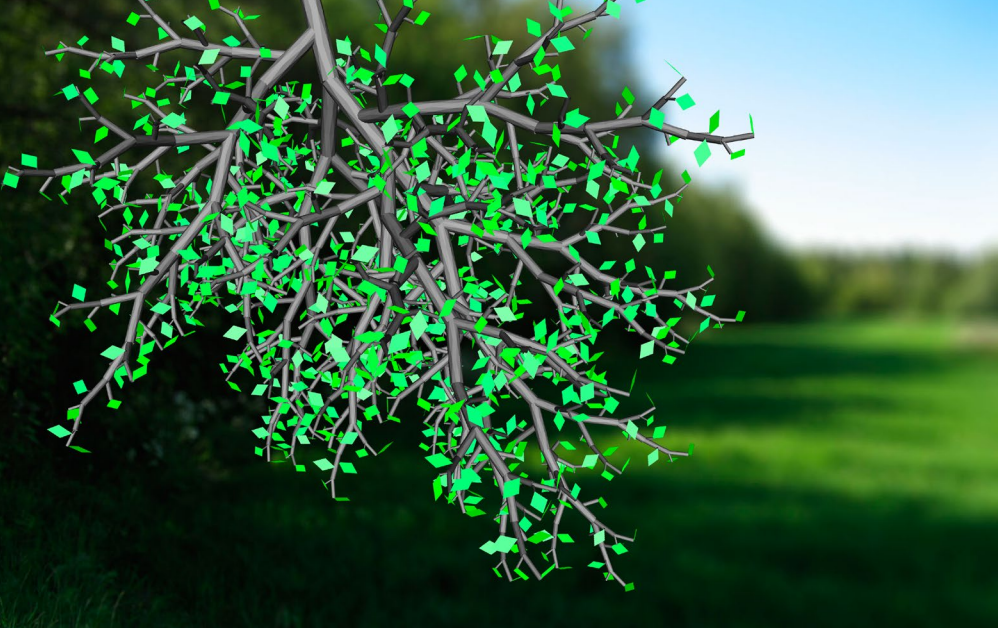
Listaus 2. Shellsort-järjestelyalgoritmi. Toteutuskieli: C.

```
Ilman oliota:

def summa(lista):
  i, tulos = 0, 0
  while i < len(lista):
    tulos = tulos + lista[i]
    i = i + 1
  return tulos

summa([1, 2, 4, 9])

Olion kanssa:

def summa(kokoelma):
  tulos = 0
  while kokoelma.available():
    tulos = tulos + kokoelma.next()
  return tulos

class ListaIteraattori:
  def __init__(s, lista):
    s.index = 0
    s.lista = lista
  def available(s):
    return s.index < len(s.lista)
  def next(s):
    cur = s.lista[s.index]
    s.index = s.index + 1
    return cur

summa(ListaIteraattori([1,2,4,9]))
```

Listaus 3. Listan summa. Olioton versio käyttää indeksejä; oliollisessa on erillinen listan läpikäyjäolio. Jälkimmäinen summa tarvitsee toimiakseen Listalteraattori-olion mutta toisaalta pystyy käyttämään mitä tahansa oliota, joka toteuttaa metodit available() ja next(). Toteutuskieli: Python.

```
Python-toteutus:

class Shape:
  def __init__(s): raise NotImplemented
  def area(s): raise NotImplemented
  def border(s): raise NotImplemented

class Circle(Shape):
  def __init__(s, radius):
    s.radius = radius
  def area(s):
    return math.pi * s.radius * s.radius
  def border(s):
    return 2 * math.pi * s.radius

myshape = Circle(3)
print "Border length is",
print myshape.border()

Scheme-toteutus:

(define (make-shape area border)
  (vector area border))
(define (shape-area shape)
  ((vector-ref shape 0)))
(define (shape-border shape)
  ((vector-ref shape 1)))

(define (make-circle radius)
  (define (circle-area) (* pi radius radius))
  (define (circle-border) (* 2 pi radius))
  (make-shape circle-area circle-border))

(define myshape (make-circle 3))
(display "Border length is ")
(display (shape-border myshape))
(newline)
```

Listaus 4. Shape-rajapinta ja sen circle-toteutus. Toteutuskieli: Python, Scheme. Python-kielessä on valmis tuki oliorakenteille, Schemessä ei.



Funktionaalinen paradigma

Funktionaalisessa paradigmassa käsitellään toisistaan riippumattomia arvoja (esimerkiksi lukuja, merkkijonoja tai listoja) sen sijaan, että muuteltaisiin yhden suuren koneen tilaa (todellisen tai virtuaalisen). Ohjelmat määritetään *funktioina*. Funktio on tarkka määritys siitä, kuinka lähtöarvoista lasketaan uusia arvoja. Funktion lopputulos riippuu ainoastaan sen saamista lähtöarvoista. Monimutkaiset funktiot määritellään yhdistelemällä yksinkertaisempia funktioita *komposition* avulla, eli määrittämällä, minkä funktion tulos annetaan millekin syötteeksi.

Intuitiivisesti voi ajatella, että funktiot ovat kuin koneita, joihin menee sisään tiettyjä arvoja (syötteitä) ja jotka tuottavat niistä toisia arvoja (tuloksia). Allegoria tosin ontuu sikäli, että funktiot eivät ”kuluta” syötettään vaan saman syötteen voi rauhassa antaa vaikka kuinka monelle funktiolle. Juuri tämä ominaisuus erottaa funktionaalisen ohjelmoinnin imperatiivisesta: imperatiivisessa ohjelmoinnissa laskennan osana usein muutellaan jo olemassa olevia muuttujia tai tietorakenteita, mutta funktionaalisessa ohjelmoinnissa vain tuotetaan uusia arvoja ja tietorakenteita vanhojen perusteella. Vanhat heitetään automaattisesti pois, kun niitä ei enää tarvita.

Funktionaalinen ohjelmointi sopii erityisen hyvin tiedon muunnoksiin ja prosessointiin. Lisäksi funktionaalinen koodi on paikallaan niissä tilanteissa, joissa koodin virheettömyys on tärkeää. Funktionaalisista ohjelmista on helpompi todistaa erilaisia ominaisuuksia, ja ne ovat yleensä jonkin verran lyhyempiä kuin vastaavat imperatiiviset ohjelmat.

Rekursio

Imperatiivisesta ohjelmoinnista tuttuja toistorakenteita ei funktionaalisessa ohjelmoinnissa voi sellaisenaan olla. Toiston päättyminen nimittäin riippuu eri muuttujien tiloista, eikä funktionaalisessa ohjelmassa muuttujan arvo enää perustamisen jälkeen muutu. Mielivaltaisen kokoisia tietorakenteita, kuten listoja, käsitellään kin funktionaalisissa ohjelmointikielissä rekursion tai siihen perustuvien funktioiden avulla. Rekursio tarkoittaa, että funktio käyttää määrittelyssään avuksi itseään (listaus 5). Rekursio on hyvin yleispätevä ohjelmointitekniikka, joka käy erityisen hyvin rekursiivisten tietorakenteiden (kuten puiden ja tiedostojärjestelmien) läpikäyntiin tai eri mahdollisuuksien tutkimiseen esimerkiksi tekoälyssä.

Tietorakenteiden muuttumattomuus ja jakaminen

Koska funktiot eivät koskaan muokkaa syötteeksi saamiaan tietorakenteita, vaan luovat aina uusia, saattaa tuntua, että funktionaalinen ohjelmointitapa kuluttaa hyvin paljon muistia. Tosiasiassa kuitenkin yhtä tietorakennetta voi käyttää toisen osana, ja monet niistä voivat jakaa keskenään osia, jos niiden muuttumattomuuteen voi luottaa. Usein on mahdollista sekä säilyttää alkuperäinen että muodostaa uusi tietorakenne hyvin pienellä muistinkulutuksella (listaus 6).

Laiskat tietorakenteet

Funktion tulos riippuu sen syötteistä eikä tilanteesta, jossa se suoritetaan. Niinpä funktionaalisessa ohjelmassa funktioiden suorittamista ja arvojen laskemista voi viivästää, kunnes tulosta todella tarvitaan. Tätä suoritusjärjestysten vapautta voi hyödyntää *laiskoissa tietorakenteissa*. Tällöin tietorakenteen sisältöä ei lasketa etukäteen, vaan siihen upotetaan funkti-

```
Iteratiivinen:          Rekursiivinen:

def summa(lista):       def summa(lista):
  tulos = 0               if len(lista) == 0:
  for e in lista:           return 0
    tulos += e            return lista[0] + \
  return tulos              summa(lista[1:])
```

Listaus 5. Listan summa. Toteutuskieli: Python.

```
Destruktiivinen:

def btree_insert(btree, value):
  if value < btree['val']:
    if btree['left'] is None:
      btree['left'] = dict(val=value,
        left = None, right=None)
    else:
      btree_insert(btree['left'],
        value)
  if value > btree['val']:
    if btree['right'] is None:
      btree['right'] = dict(val=value,
        left = None, right=None)
    else:
      btree_insert(btree['right'],
        value)

Applikatiivinen:

def btree_insert(btree, value):
  if btree is None:
    return (value, None, None)
  curvalue, left, right = btree
  if value < curvalue:
    return (curvalue,
      btree_insert(left, value), right)
  if value > curvalue:
    return (curvalue,
      left, btree_insert(right, value))
  return btree
```

Listaus 6. Uuden arvon sijoitus järjestettyyn binääripuuhun. Destruktiivinen versio muokkaa alkuperäistä tietorakennetta, applikatiivinen säilyttää sen. Toteutuskieli: Python.

```
Python:

def delay(fn):
  return ['delay', fn]

def force(thunk):
  if thunk[0] == 'delay':
    thunk[0] = 'val'
    thunk[1] = thunk[1]()
  return thunk[1]

def numbers_from(n):
  return delay(lambda:
    (n, numbers_from(n+1)))

naturals = numbers_from(0)

def double(stream):
  n, rest = force(stream)
  return delay(lambda:
    (n*2, double(rest)))

evens = double(naturals)

Scheme:

(define (numbers-from n)
  (delay (cons n (numbers-from (+ n 1)))))

(define naturals (numbers-from 0))

(define (double stream)
  (let ((val (force stream)))
    (delay (cons (* 2 (car val))
                 (double (cdr val))))))

(define evens (double naturals))
```

Listaus 7. Luonnolliset luvut ja parilliset luvut laiskana listana. Toteutuskieli: Python, Scheme. Scheme sisältää valmiiksi laiskojen tietorakenteiden käsittelyä helpottavia palveluita, Python ei.

oita, joilta voi tarvittaessa pyytää rakenteen varsinaisen sisällön.

Laiskat tietorakenteet voivat olla äärettömän suuria, koska niitä lasketaan vain tarpeen mukaan. Esimerkkejä laiskoista tietorakenteista ovat muun muassa laiskat listat eli virrat tai vaikkapa puutietorakenne, josta löytyy jokainen mahdollinen shakkipelin kulku. Monet algoritmit ovat esitettävissä elegantisti muunnoksina, joissa funktio lukee yhtä laiskaa tietorakennetta ja tuottaa sen perusteella toista (listaus 7).[2] Laiskat tietorakenteet sopivat lukuisiin tarkoituksiin aina numeerisista menetelmistä optimointiin ja jäsentämiseen.[3]

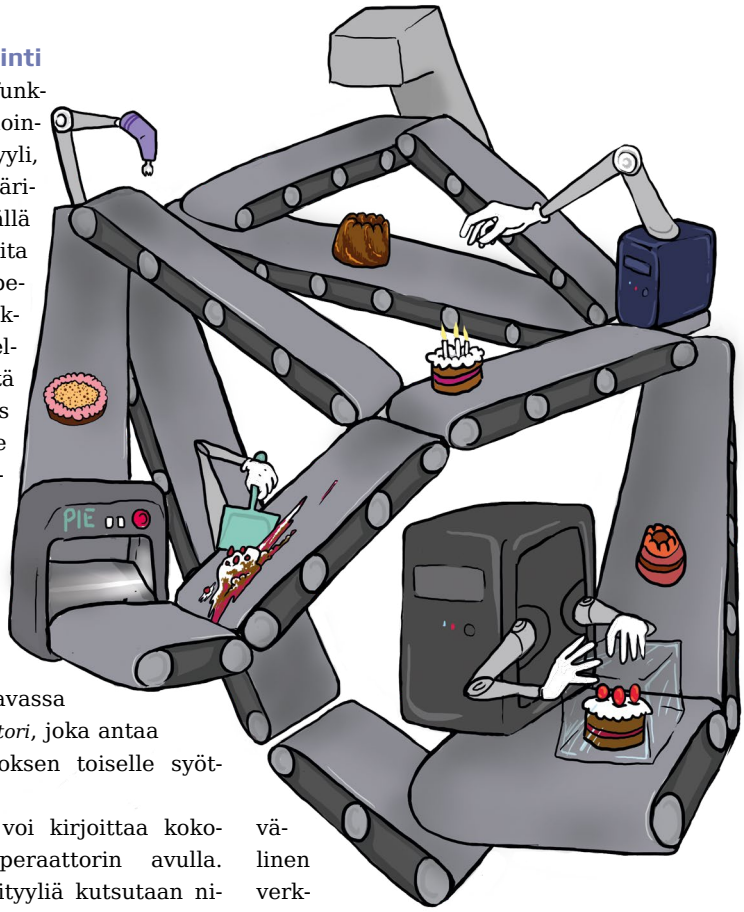
Korkeamman kertaluvun funktiot

Yleensä funktionaaliset ohjelmointikielet tukevat myös korkeamman kertaluvun funktioita eli *funktionaaleja* sekä niiden käyttökelpoisuutta huimasti lisääviä *sulkeumia (closure)*. Funktionaali tarkoittaa funktiota, joka ottaa syötteekseen tai palauttaa toisia funktioita. Rekursio on usein korvattavissa sopivan funktionaalin käytöllä (listaus 8). Funktionaalit soveltuvat erittäin monenlaisten toimintatapojen, tietorakenteiden ja laskentaympäristöjen abstrahointiin.[4]

Monessa tilanteessa funktionaalien käyttö toimii vaihtoehtona olio-ohjelmoinnille. Olio-ohjelmoinnissahan pystyy kirjoittamaan tietorakenteesta riippumattomia algoritmeja käyttämällä tietorakenteiden tarjoamia metodeja. Funktionaalit taas saavat argumentteina ne funktiot (aliohjelmat), joilla rakennetta käsitellään. Molemmilla tavoilla saadaan aikaiseksi tietotyyppien suhteen yleistettyä koodia (listaus 9).

Tietovuo-ohjelmointi

90-luvulta lähtien funktionaalisessa ohjelmoinnissa on yleistynyt tyyli, jossa funktiot määritellään yhdistelemällä funktionaaleilla muita funktioita. Ero perinteisempään funktionaaliseen ohjelmointiin on se, että arvojen kuljetus funktiolta toiselle on annettu funktionaalien tehtäväksi eikä arvoihin viitata enää niin paljon erillisillä muuttujilla. Erityisen keskeinen funktionaali tässä ohjelmointitavassa on *kompositio-operaattori*, joka antaa yhden funktion tuloksen toiselle syötteeksi.[5]



Monet funktiot voi kirjoittaa kokonaan kompositio-operaattorin avulla. Tällaista ohjelmointityyliä kutsutaan nimellä *point-free notation*, ja se tuo usein selkeästi esiin tietovuot, joiden kautta arvot kulkevat laskennassa (listaus 10). Koska kaikki funktiot voi määritellä yksinkertaisempien funktioiden kompositioina, *konkatenatiivisissa* ohjelmointikielissä kompositiosta tehdään ohjelmoinnin perusoperaatio.[6]

Tietovuo-ohjelmointi tuottaa kaunista ja selkeää koodia silloin, kun funktioiden kompositiorakenne ei ole tavattoman monimutkainen. Kun funktioiden välinen verkko monimutkaistuu, tarvitaan myös koko ajan lisää funktionaaleja ja muita apukeinoja syötteiden ohjaamiseksi oikeille funktioille.

```
Rekursiivinen:

def summa(lista):
  if len(lista) == 0:
    return 0
  return lista[0] + \
    summa(lista[1:])

def toistoton(lista):
  if len(lista) <= 1:
    return lista
  if lista[0] == lista[1]:
    return toistoton(lista[1:])
  return lista[0] +
toistoton(lista[1:])

Funktionaaleihin perustuva:

def summa(lista):
  return reduce(lambda x,y: x+y,
                lista, 0)

def toistoton(lista):
  return map(lambda (x,y): x,
   filter(lambda (x,y): x != y,
     zip(lista, lista[1:]))) +
lista[-1:]
```

Listaus 8. Listan summa ja peräkkäisten toistuvien elementtien poisto. Toteutuskieli: Python.

```
Metodin avulla:

def is_ordered(ls):
  for e1, e2 in zip(ls, ls[1:]):
    if not e1.__lt__(e2):
      return False
  return True

Argumentin avulla:

def is_ordered(ls, is_less):
  for e1, e2 in zip(ls, ls[1:]):
    if not is_less(e1, e2):
      return False
  return True
```

Listaus 9. Onko lista järjestyksessä? Alkioita vertaillaan metodilla tai argumenttina saadulla funktiolla. Toteutuskieli: Python.

```
Rekursiivinen:

countEvens [] = 0
countEvens (x:xs)
  | even x = 1 + countEvens xs
  | otherwise = countEvens xs

Tietorakennefunktionaaleilla:
countEvens xs = length (filter
even xs)

Kompositio-operaattorilla:
countEvens = length . filter even
```

Listaus 10. Parillisten laskeminen. Toteutuskieli: Haskell.

[2] Tietyissä ohjelmointikielissä, kuten Haskellissa, kaikki tietorakenteet ovat laiskoja.

[3] Niillä voi myös faktoroida lopetusehdot erilleen iteratiivisista algoritmeista siten, että iteratiivinen algoritmi tuottaa äärettömästi tarkentuvan listan tuloksia ja listaa lukeva funktio päättää, milloin on aika lopettaa.

[4] Korkeamman kertaluvun funktioiden käyttö on hyödyllistä muutenkin kuin funktionaalisen ohjelmoinnin kannalta. Imperatiivisessa ohjelmassa koodia syötteekseen ottavista aliohjelmista on lähes yhtä paljon hyötyä kuin funktionaalisessa ohjelmassa. Joidenkin tunnettujen imperatiivisten kielten (Tcl, Ruby ja Smalltalk) ehto- ja toistorakenteita on toteutettu tällaisilla menetelmillä.

[5] Muita tärkeitä funktionaaleja ovat *monadisen* ohjelmoinnin bind-toteutukset sekä tietorakenteita läpikäyvät *katamorfismit*, kuten listojen reduce().

[6] Tunnetuin näistä kielistä on Factor. Monelle tietovoita korostava point-free notation on tutumpi Unix-komentoriviympäristön *putkirakenteista*. Esimerkiksi seuraava komento laskee syötteensä sanojen frekvenssijakauman:

```
tr ' ' \\012 | sort | uniq -c | sort -nr
```

Logiikkaparadigma

Logiikkaparadigmassa laskentaa tarkastellaan prosessina, jossa etsitään ehtoja, joilla annetut väitteet pitävät paikkansa. Esimerkiksi väite [1, X, 3] = [1, 2, Y] pitää paikkansa silloin ja vain silloin, kun X = 2 ja Y = 3. Ohjelmia muodostetaan määrittämällä *predikaatteja*. Predikaatti on sääntö, joka määrittää, missä suhteessa sen *argumentit* ovat toisiinsa.

Monimutkaiset predikaatit määritellään yleensä sen perusteella, miten ne riippuvat yksinkertaisemmista (listaus 11). Laskenta etenee siten, että logiikkalausekkeita seuraamalla etsitään ne muuttujien arvot, joilla kaikki ohjelman väitteet pitävät yhtä aikaa paikkansa. Logiikkaohjelmointi soveltuu erityisen hyvin tilanteisiin, joissa etsitään ratkaisua monimutkaiseen rajoiteongelmaan (esimerkiksi liikennevalojen ajastukseen), haetaan erilaisista vaihtoehdoista parasta (esimerkiksi shakkitekoäly) tai parseroidaan eli jäsennetään kieliä.

Listauksen käyttöesimerkeistä näkyy, että määritelmien perusteella voi laskea pituuksia listoista, listoja pituuksista, yhdistettyjä listoja osalistoista tai osalistoja yhdistettävistä listoista. Lisäksi ratkaisuja on joskus yksi, joskus monta ja joskus ei yhtään.

Sekä logiikka- että funktionaalinen paradigma ovat esimerkkejä *deklaratiivisesta* ohjelmoinnista. Deklaratiiviset ohjelmat eivät ole vain ohjeita siitä, miten jotain lasketaan, vaan ne voi tulkita myös väitteiksi siitä, miten asiat ovat. Funktioiden määrittelyt ovat käytännössä väitteitä siitä, mikä on funktion syötteiden ja tuloksen suhde. Esimerkiksi seuraava Haskell-ohjelma kertoo, mikä on listan ja sen pituuden suhde.

```
length [] = 0
length (x:xs) = 1 + length xs
```

Logiikkaohjelmointi menee tässä pidemmälle ja tekee ohjelman predikaateista *symmetrisiä* siten, että niissä ei määritetä, mikä on syötettä ja mikä tulosta, vaan väitteen yhden argumentin voi laskea toisen perusteella ja päinvastoin.

Indeterminismi

Kuten todettu, logiikkaohjelmassa asetettuihin ehtoihin voi olla yksi, monta tai ei yhtään ratkaisua. Joskus ohjelmaa suoritettaessa tiettyyn osaongelmaan on monta ratkaisua, mutta jokin niistä ei myöhempien sääntöjen vuoksi kelpaakaan. Tällöin joudutaan palaamaan takaisin (backtrack) muihin mahdollisiin ratkaisuihin ja katsomaan, tyydyttäisikö seuraava niistä loputkin rajoitteet. Tällaista suoritustapaa, jossa mahdollisia tuloksia on useita ja kaikki mahdollisuudet tutkitaan läpi, kutsutaan *indeterministiseksi* (listaus 12). Indeterminismi ei ole ainoastaan logiikkaohjelmointiin rajoittuva tekniikka, vaan myös funktionaalisessa ohjelmoinnissa voi toteuttaa funktioita, jotka palauttavat haluttaessa useampia arvoja (tai ei yhtään).[7]

```
Listan pituus:

length([], 0).
length([F|Rest], X) :-
  length(Rest, Y),
  plus(Y, 1, X).

Käyttöesimerkit:

?- len([4,6,2], X).
X = 3 .
?- len([1,2|X], 4).
X = [_G335, _G338] .
?- length([1,X,3], 2).
false.

Listojen katenaatio:

cat([], L2, L2).
cat([E1|L1], L2, [E1|L3])
:-
  cat(L1, L2, L3).

Käyttöesimerkit:

?- cat([1,2], [3,4], X).
X = [1, 2, 3, 4].
?- cat(X, Y, [a, b]).
X = [],
Y = [a, b] ;
X = [a],
Y = [b] ;
X = [a, b],
Y = [] .
```

Listaus 11. Listan pituus ja kahden listan yhdistäminen. Toteutuskieli: Prolog.

Oppia ikä kaikki

Tässä jutussa on käsitelty vain pintapuolisesti kaikkein yleisimpiä niistä paradigmoista, joiden avulla ohjelman toimintatavan voi määritellä. Jokainen esitellyistä ohjelmointiparadigmoista kantaa mukanaan vuosikymmenten perinnettä ja tutkimusta siitä, miten erilaiset ongelmat voi ratkaista elegantisti. On kuitenkin olemassa myös uudempia paradigmoja, joiden tutkimus on vasta alullaan. Jo vähäinenkin ohjelmointitapojen tuntemus kehittää ohjelmoijan taitoja, mutta lopulta erilaiset ohjelmointiparadigmat tarjoavat kiinnostuneille lähes ehtymättömän aarreaitan.

Toivon, että tämä kirjoitus rohkaisee lukijoita tutustumaan uusiin ohjelmointitapoihin. Ohjelmointi on ajattelua, joten uudet ohjelmointitavat ovat samalla uusia ajatustapoja ja auttavat parhaimmillaan ymmärtämään myös maailmaa paremmin.

[7] Puhtaissa funktionaalisissa kielissä indeterministinen funktio yleensä palauttaa laiskan listan erilaisista tuloksistaan, mutta jatkeita (*continuations*) tukevissa ohjelmointikielissä indeterminismin voi toteuttaa myös hyppimällä jatkeilla ohjelman suorituksessa takaisin aikaisempiin vaiheisiin.

Kehitys ei kehity

Mikko Heinonen

Ohjelmiston päivittämisestä saisi olla jotain hyötyäkin.

Oletteko koskaan miettineet, mistä syntyy se vanhojen ohjelmistojen kiviriippa, joka vaikeuttaa vaikkapa eri käyttöjärjestelmään siirtymistä yrityksessä? Miksi ihmeessä osa käyttäjistä ei halua päivittää ohjelmiaan uuteen versioon?

Minulle aihe on hyvin tuttu. Työssäni teknisenä kääntäjänä joudun päivittäin käyttämään ohjelmia, joista moni tietokoneen käyttäjä ei ole kuullutkaan. Dokumentteja, sopimustekstejä ja ohjelmistoja käännetään lause tai sana kerrallaan käännösmuistiohjelmilla, jotka pyrkivät nopeuttamaan kääntäjän työtä tallentamalla vanhat käännökset tietokantaan ja ehdottamalla käännöksiä. Hyvin toimiessaan järjestelmä vauhdittaakin työtä reippaasti.

Valitettavasti hyvin toimiminen on nykyisin lähinnä harvinainen poikkeus. Trados, ohjelmista tunnetuin, käyttää apunaan Microsoft Wordin lisäosaa, joka on ilmeisesti kirjoitettu jossain vuoden 2000 tienoilla. Asennus nykyisiin Wordin versioihin edellyttää tiedostojen manuaalista kopiointia ja uudelleen nimeämistä, eikä se välttämättä toimi silloinkaan.

Saman ohjelman uudempi versio, Trados Studio, poisti Word-symbioosin ja toimii uudemmissakin käyttöjärjestelmissä hyvin. Samalla se on kuitenkin kadottanut edeltäjänsä erinomaiset muistihakutoiminnot ja tehnyt käytöstä niin monimutkaista, että lyhyenkin tiedoston kääntäminen vaatii useita projektinhallintaoperaatioita.

Seurauksena on se, että useat käännöstoimistot lähettävät töitä vielä tänäkin päivänä vanhalla Tradosilla tehtäväksi. Tämä siitäkin huolimatta, että se sai viimeisen päivityksensä vuonna 2007 ja omistajayhtiö on yrittänyt aktiivisesti tappaa sitä myyntinsä edistämiseksi.

Hymyä kyynelten läpi

Tuo vanha sotaratsu onkin edelleen kenties paras työkalu, kun on vain tarkoitus kääntää pikaisesti tekstiä. Se on paras, vaikka se usein pilaa asiakirjan muotoiluja ja saattaa yhtäkkiä kieltäytyä siirtymästä seuraavaan lauseeseen. Se on paras siitäkin huolimatta, että ohjelman ikkuna voi useampaa näyttöä käytettäessä siirtyä yhtäkkiä X-koordinaattiin 10 000.

Varsinaisen työnkulun kannalta Trados tuntuu nimittäin olevan niitä harvoja kääntäjän näkökulmasta kirjoitettuja ohjelmia. Nykyisin myynnissä olevat työkalut ovat pääosin insinöörihirvityksiä, joissa on kymmenittäin valikoita ja asetuksia. Ne kyllä osaavat satoja eri tiedostomuotoja, mutta käyttäjäystävällisyys ei kuulu niiden sanavarastoon. Eikä ole salaisuus, että kääntäjinä työskentelee paljon ihmisiä, joiden harrastuneisuus ei riitä sekavien valikoiden kaiveluun.

Tradosin sitkeyttä selittäneekin osaltaan se, että sen käyttöliittymä on käytännössä Microsoft Word. Toimistosovellusten monopolista voi toki olla käärmeissään, mutta Wordin käytettävyys on kyllä kohdallaan verrattuna keskimääräiseen, lähinnä taulukkolaskennalta näyttävään käännösohjelmaan. Se on tehty kirjoitetun kielen käsittelyyn, ei merkkijonojen koodaukseen.

> *Ohjelmistoja kehitetään epäolennaisiin suuntiin ja niistä tehdään hitaampia.*

Ehkä juuri tämän vuoksi yksikään uudemmista käännösohjelmistoista ei ole onnistunut nousemaan uudeksi de facto -standardiksi, vaan alan suuret toimijat ovat hankkineet kuka minkäkin ohjelmiston käyttöönsä. Ne eivät ole juurikaan yhteensopivia, joten freelance-kääntäjän tai suurten toimistojen alihankkijayrityksen on ylläpidettävä useita eri lisenssejä voidakseen ottaa töitä vastaan. Yksittäinen lisenssi voi maksaa tuhatkin euroa. Tonni on aika iso raha ohjelmasta, joka ei tee juuri mitään sen paremmin kuin vanha Trados.

Suunta on väärä, vauhtia liikaa

Minun on mahdotonta uskoa, ettei kunnollista käännösohjelmaa saisi aikaan. Itse asiassa omaan käyttööni sellainen olisi jo olemassa, jos saisin yhdistellä nykyisten ominaisuuksia vapaasti. Kukaan ei vain kysy asiaa kääntäjiltä, vaan ohjelmistot kirjoitetaan etupäässä projektinhallinnan käyttöön. On yhteistyötä, konekäännöstä ja online-työnkulkua, mutta ei edelleenkään hyvää muistihakua, vaikka se olisi itse työn kannalta huomattavasti oleellisempaa.

Kohtuullisen pieni käännösohjelma-ala tuntuu vetävän puoleensa pyörän uudelleen keksijöitä, jotka kieltäytyvät matkimasta ominaisuuksia toisiltaan. Lähestymistavasta on tuloksena joukko työkaluja, jotka toistavat toistensa virheitä ja hukkaavat hyvät ominaisuutensa huonojen joukkoon.

Minun on siis hyvin helppo ymmärtää, miten syntyvät ne historialliset ohjelmistokertymät, jotka aiheuttavat ongelmia päivitysten yhteydessä. Pienille aloille ei tunnu riittävän ohjelmistokehityksen osaamista, joka saisi pidettyä viisasten kiven hallussaan sen kerran löydettyään. Ohjelmistoja kehitetään epäolennaisiin suuntiin ja niistä tehdään yleensä entistä hitaampia. Tällöin käyttäjä, joka haluaa vain tehdä työtään, kahlitsee itsensä vanhaan luottosoftaansa kuin luontoaktivisti puunrunkoon: tätä ette minulta vie.

Common Lisp
– ohjelmoitava ohjelmointikieli

Common Lisp on samanaikaisesti tietojenkäsittelytieteen klassikko ja nykyaikainen ohjelmointikieli. Lisp-kielten evoluutio on rakentanut vanhojen perusideoiden varaan uuden kielen, joka vastaa nykyajan tarpeisiin ja haastaa ilmaisuvoimallaan kokeneenkin ohjelmoijan.

Teksti: Teemu Likonen
Kuvat: Risto Mäki-Petäys, Marcin Wichary, Roland Dobbins, Teemu Likonen, Dave Fischer

Sukujuuret syvällä

Tässä artikkelissa keskityn kieleen nimeltä Common Lisp, joka on vain yksi murre Lispin suuressa perheessä. Common Lisp on uudehko perheenjäsen, mutta kieliperheen juuret ovat syvällä historiassa.

Lisp-kielen perusteet loi John McCarthy vuonna 1956. Tuolloin kehitystyö kytkeytyi keinoälytutkimukseen IBM:llä, Darmouthin yliopistolla sekä Massachusettsin teknillisessä korkeakoulussa. Uuden kielen varsinainen tietokonetoteutus alkoi syksyllä 1958, ja sitä pidetäänkin kielen syntymävuotena. Kielen nimi oli LISP, joka on lyhenne sanoista ”list processing”.

Useat toisistaan riippumattomat tahot innostuivat kielen kehittämisestä, ja pian se jakautui useiksi eri murteiksi. Uusia ominaisuuksia lisättiin, ja alkuperäinen lyhennenimi ”listojen käsittelijä” alkoi jäädä syrjään. Kieliperheen nimeksi vakiintui Lisp.

1960-luvun lopussa oli jo olemassa tai kehitteillä MacLisp, ZetaLisp ja Interlisp. Kohta mukaan tulivat myös Portable Standard Lisp, NIL, S-1 Lisp, Franz Lisp, Scheme ja SpiceLisp. Lisp-kielisten ohjelmien tehokasta suorittamista varten rakennettiin 1970- ja 1980-luvulla erityisiä tietokoneitakin, Lisp-koneita. Niiden käyttöjärjestelmät ohjelmoitiin tietysti Lispillä.

Melko nuori perheenjäsen on Common Lisp, joka syntyi yhdistämään aiempia Lispin murteita ja hajanaista kehitystyötä. Haluttiin määritellä standardi Lisp, jossa yhdistyvät monien aiempien murteiden ominaisuudet. Kieli sai virallisen aseman vuonna 1994, kun siitä tuli yhdysvaltalainen ANSI-järjestön standardi.

Luonteenpiirteitä

Common Lisp on yleiskäyttöinen dynaaminen kieli, joka tukee useita ohjelmointityylejä. Se on tarkoitettu käytännöllisten ohjelmien kirjoittamiseen ja todellisten ongelmien ratkaisemiseen. Se ei ole akateeminen kielentutkimushanke eikä pedagoginen esimerkkikieli.

Dynaamisen luonteensa vuoksi koko kieli on läsnä

Symbolicsin Lisp-kone Googlen museossa.

John McCarthy (1927–2011), Lispin keksijä.

```
(defgeneric handle-input-message (client message))

(defmethod handle-input-message ((client client) (message server-privmsg-eval))
  (let ((target (first (arguments message)))
        (user (trivial-irc:prefix-nickname (prefix message)))
        (contents (subseq (second (arguments message))
                          (length *eval-prefix*)))
        (sandbox-name (common:user-to-sandbox-name (prefix message))))

    (send :terminal message)
    (sandbox-init sandbox-name)

    (with-thread ("eval and print" :timeout *eval-timeout*)
      (handler-case
          (let ((string (clean-string (with-output-to-string (stream)
                                        (sandbox-repl sandbox-name
                                                      contents stream)))))
            (when (plusp (length string))
              (let ((msg (make-instance 'client-privmsg :target target
                                        :contents string)))
                (send :terminal msg)
                (queue-add (send-queue client) msg))))

        (common:extra-command (c)
          (with-thread ("extra command")
            (let ((cmd (common:command c))
                  (args (common:arguments c)))
              (cond ((equalp cmd "help")
                     (extra-cmd-help client target))
                    ((equalp cmd "tell")
                     (extra-cmd-tell client user args)))))))

      :timeout
      (let ((msg (make-instance 'client-privmsg :target target
                                :contents (bot-comment "EVAL-TIMEOUT"))))
        (send :terminal msg)
        (queue-add (send-queue client) msg)))))
```

Lisp-koodi koostuu sisäkkäisistä listarakenteista.

ohjelman suorituksen aikana. Ohjelma voi esimerkiksi luoda suorituksen aikana uusia funktioita tai määritellä vanhoja uudelleen. Funktioita voi myös kääntää ohjelman suorituksen aikana, niin että niiden suoritus nopeutuu.

Myös muuttujien tyypit ovat dynaamisesti muutettavissa, eikä niitä tarvitse julistaa tietyn tyyppiseksi. Standardi silti määrittelee declare-lausekkeen, jonka avulla kääntäjälle voi kertoa, minkä tyyppisiä muuttujat ovat. Se auttaa optimoinnissa, ja tehokkaimmat Common Lisp -kääntäjät tuottavatkin suoritusteholtaan lähes C-kielen kääntäjien tasoista konekieltä.

Useissa ohjelmointikielissä ovat erikseen lauseet ja lausekkeet. Karkeasti ilmaistuna lauseet tekevät jotakin ja lausekkeet palauttavat jonkin arvon. Common Lispissä on pelkästään lausekkeita eli kaikki ohjelmarakenteet palauttavat arvon. Tämä ominaisuus kytkeytyy sujuvasti funktionaaliseen ohjelmointiin, koska palautettuja arvoja voi hyödyntää tehokkaammin.

Common Lisp on myös ohjelmoitava ohjelmointikieli, mikä tarkoittaa, että kieltä itseään voi muokata. Ohjelmoija voi lisätä kieleen uusia ominaisuuksia ja syntaktisia rakenteita.

> *We were not out to win over the Lisp programmers; we were after the C++ programmers. We managed to drag a lot of them about halfway to Lisp.” – Guy Steele, yksi Java-kielen määritelmän kirjoittajista*
>
> lähde: http://people.csail.mit.edu/gregs/ll1-discuss-archive-html/msg04045.html

Sulkeiden sekaan

Leikillisesti on sanottu LISP-lyhenteen tarkoittavan ”lots of irritating superfluous parentheses” tai muuta vastaavaa. Päällisin puolin Lisp-ohjelmassa näyttääkin olevan hyvin paljon sulkeita. Leikillisestä selityksestä huolimatta sulkeet eivät ole liiallisia eivätkä tarpeettomia.

Lispin merkittävin syntaktinen ominaisuus on se, että ohjelmakoodi muodostuu kielen omista tietorakenteista. Sulkeita on paljon, koska niiden avulla merkitään tietotyyppi nimeltä lista. Listoja käytetään paitsi tietorakenteena mutta myös funktioiden, makrojen ja muiden ohjelmarakenteiden esittämiseen.

Jos lista suoritetaan funktiokutsuna, sen ensimmäinen elementti on funktion nimi ja loput elementit argumentteja. Funktiokutsu näyttää esimerkiksi tältä: (+ 1 2 3). Nimi ”+” viittaa yhteenlaskufunktioon. Muutkin ohjelman rakenteet ilmaistaan listojen avulla. Seuraava esimerkki luo yksinkertaisen silmukan dotimes-makron avulla.

```
(dotimes (i 10)
  (write-line "Tätä 10 kertaa!"))
```

Lisp-ohjelma voi näyttää monimutkaiselta, jos kieli on lukijalle outo. Loogisesti kielen syntaksi on kuitenkin erittäin yksinkertainen, koska ohjelmakoodin tulkitsemisen säännöt ovat yksinkertaiset ja koska ohjelmakoodi muodostuu kielen omista tietorakenteista. Yhdessä nämä ominaisuudet tekevät metaohjelmoinnista helppoa. Lisp-ohjelman on hyvin helppoa lukea, muokata ja tuottaa uutta Lisp-koodia, ja siksi Lisp-kieliä sanotaankin ohjelmoitaviksi ohjelmointikieliksi.

Tiedon tyypit

Common Lispissä on suunnilleen samat tietotyypit kuin muissakin korkeatasoisissa kielissä. Niitä ovat esimerkiksi kokonaisluku, liukuluku, merkkijono, lista, taulukko, hajautustaulu ja luokka. Tässä yhteydessä esittelen harvinaisempia ominaisuuksia.

Luku-tietotyypeissä on eräitä erikoisuuksia moniin ohjelmointikieliin verrattuna. Yksi hienous ovat rationaaliluvut. Niiden ansiosta lukuja voi käsitellä mielivaltaisen tarkasti ilman pyöristysvirheitä. Esimerkiksi jakolaskusta (/ 4 3) saadaan tulokseksi rationaaliluku 4/3 eikä pyöristettyä liukulukua. Luvuista mainittakoon myös, että ne voivat olla mielivaltaisen suuria. Lisäksi Common Lisp tukee kompleksilukuja.

Perinteisesti ohjelmointikielissä merkkijono ei todellisuudessa ole merkkien jono vaan lukujono. Luvut vain tulkitaan merkkien koodeiksi jonkin merkistökoodauksen avulla. Common Lispissä myös merkki on oma tietotyyppinsä, jolla on oma syntaksinsa. Ohjelmoijan kannalta merkki ei ole sidoksissa sen tekniseen toteutustapaan tietokoneella. Nykyaikana Common Lispin toteutukset noudattavat Unicode-merkistöä ja sen koodaustapoja.

Common Lispin merkkijono on vektori eli yksiulotteinen taulukko, jonka kaikki elementit ovat merkkejä. Tyyppijärjestelmässä merkkijono on vain taulukon erikoistapaus, ja sitä voi käsitellä samoilla funktioilla kuin taulukoitakin.

Nykyaikana on varsin tavallista, että funktiot ovat ensiluokkaisia. Ominaisuus tarkoittaa esimerkiksi sitä, että funktioita voi luoda ohjelman suorituksen aikana, tallentaa muuttujiin tai muihin tietorakenteisiin ja antaa argumentiksi toisille funktioille. Tämä ominaisuus toteutettiin ensimmäisenä nimenomaan Lispissä, ja se on mukana myös Common Lispissä. Standardikirjastoon kuuluu useita niin sanottuja korkeamman kertaluvun funktioita, jotka tarvitsevat argumentikseen toisen funktion. Kun funktio määritellään, se muodostaa sulkeuman (engl. closure), jossa voi viitata sen ulkopuolella määriteltyihin muuttujiin.

Symbolics 3600 -Lisp-koneen näppäimistö.

Virta on tiedonsiirtoon tarkoitettu tietotyyppi. Virrasta voidaan lukea tietoa tai siihen voidaan kirjoittaa tietoa. Virran elementit ovat merkkejä (merkkivirta) tai tavuja (binaarivirta). Tiedostojen lukeminen ja kirjoittaminen on käytännössä virtojen käsittelyä. Tiedosto voidaan avata esimerkiksi merkkivirraksi ja syntyneestä virtaobjektista voidaan lukea merkkejä. Myös verkkoliikenne ja tekstipohjainen viestintä ohjelman käyttäjän kanssa tapahtuvat virtojen avulla. Ohjelmoija voi luoda virtoja omiinkin tarkoituksiinsa, esimerkiksi ohjelman sisäiseen viestintään.

Muita tietotyyppien erikoisuuksia ovat tiedostopolut ja symbolit. Tiedostopolku sisältää tietoa, joka nimeää tiedoston tiedostojärjestelmässä. Symboleja puolestaan käytetään yksilöllisinä tunnisteina ja niminä. Ne nimeävät esimerkiksi muuttujia, funktioita, makroja, tiloja ja luokkia. Symboleja voi luoda omiin tarkoituksiinkin, yksilölliseksi tunnisteeksi mille tahansa.

Valikoituja ominaisuuksia

Monessa mielessä Common Lisp on hyvin samanlainen kuin nykyaikaiset korkeatasoiset dynaamiset kielet. Lisp onkin ollut innoittajana monille kielille. Samankaltaisuudesta huolimatta erityispiirteitäkin on. Esittelen seuraavaksi ominaisuuksia, jotka ovat joko itsessään harvinaisuuksia tai joiden toteutus on poikkeuksellinen.

Makrot

Makrot ovat niin sanottuja metaohjelmia, koska ne tuottavat ohjelmakoodia ohjelmallisesti. Niiden avulla luodaan syntaktisia abstraktioita eli muokataan itse ohjelmointikieltä. Makrot ovat kiinteä osa Common Lispiä, ja monet kielen valmiista ominaisuuksista on toteutettu makrojen avulla. Ohjelmoija voi määritellä niitä myös itse. Jos metaohjelmointia ei ole ennen harrastanut, voi olla aluksi hankalaa ymmärtää makron ja funktion eroa. Lispissä molemmat nimittäin näyttävät samalta.

Funktiot toimivat ohjelman suorituksen aikana. Ne niputtavat ohjelmakoodia yhteen, suorittavat määrätyn tehtävän ja palauttavat jonkin arvon. Makrot puolestaan muokkaavat ohjelmakoodia toisenlaiseksi. Makro tuottaa ohjelmakoodia ennen kuin ohjelmaa varsinaisesti vielä suoritetaan. Ohjelman suorituksen aikana ei makroja enää ole, vaan ainoastaan niiden tuottama koodi.

Esimerkin vuoksi luodaan yksinkertainen makro. Ohjelmoija kirjoittaa makron usein siksi, että hän haluaa välttää samankaltaisten rakenteiden toistuvaa kirjoittamista. Oletetaan, että seuraavanlainen rakenne toistuu koodissa usein:

```
;; Asetetaan muuttujalle funktiokutsun arvo:
(let ((arvo (funktiokutsu)))
  ;; Testataan, onko arvo tosi:
  (if arvo
      ;; On tosi. Käytetään muuttujaa:
      (tee-jotain arvo)
      ;; Epätosi. Tehdään jotain muuta:
      (tee-muuta)))
```

Edellisessä esimerkissä luodaan väliaikainen muuttuja ja asetetaan sille funktiokutsun arvo. Sen jälkeen käytetään arvoa if-rakenteen ehtolausekkeessa ja vielä toisessa if-rakenteen haarassa. Kun tällaista täytyy kirjoittaa usein, toivoisi ehkä kieleen seuraavanlaista rakennetta:

```
(if-let (arvo (funktiokutsu))
        (tee-jotain arvo)
        (tee-muuta))
```

Näin väliaikainen muuttuja määriteltäisiin osana ehtolauseketta ja ohjelmoija säästyisi hieman vähemmällä kirjoittamisella. Onneksi on helppoa kirjoittaa makro, joka luo kyseisen if-let-rakenteen. Common Lispissä makrot määritellään defmacro-nimisellä makrolla ja ne kirjoitetaan tavallisella Common Lispillä. Kieli on siis samalla itsensä metakieli.

```
(defmacro if-let ((muuttuja ehto)
                  tosi &optional epätosi)
  `(let ((,muuttuja ,ehto))
     (if ,muuttuja ,tosi ,epätosi)))
```

Olemme luoneet yksinkertaisen makron eli syntaktisen abstraktion. Uusi if-let-rakenne helpottaa ohjelmoijan työtä, koska se tuottaa ohjelmallisesti koodia, jonka

toistuva kirjoittaminen on tylsää.

Vaikka edellä esitelty makro onkin käyttökelpoinen, se ei anna kattavaa kuvaa makrojen todellisista kyvyistä. Makrojen avulla voi lopulta luoda ikään kuin pienoiskielen, jota ohjelmoija sitten käyttää oman ohjelmansa kirjoittamiseen. Tällä tavoin abstrahoidaan ensin kielen syntaksia ja käytetään sitten tätä korkeammantasoista kieltä varsinaisen ohjelman kirjoittamiseen.

Silmukkatyökalu LOOP

Makrot antavat ohjelmoijalle mahdollisuuden kehittää kielen syntaksia, mutta myös monet Common Lispin valmiit ominaisuudet on toteutettu makrojen avulla. Yksi tällainen on loop, joka sisältää silmukoihin erikoistuneen ohjauskielen.

loop-makro lienee syntynyt ajatuksesta, että monimutkaistenkin silmukkarakenteiden kirjoittamisen pitäisi olla mahdollisimman helppoa ja että usein toistuva rutiinikoodi pitää abstrahoida pois näkyvistä. loop-rakenteessa olevilla avainsanoilla säädellään silmukan toimintaa. Avainsanojen lomassa on tavallista Lisp-koodia. Seuraava esimerkki lienee suunnilleen ymmärrettävissä, vaikkei makroa ennestään tuntisikaan:

```
(loop with käsitellyt = (make-array 10)
      for numero from 0 upto 9
      for rivi = (read-line virta nil)
      for käsitelty = (käsittele rivi numero)
      while rivi
      collect rivi into rivit
      do (setf (elt käsitellyt numero) käsitelty)
      finally (return (values rivit käsitellyt)))
```

Yleiskäyttöinen SETF

setf on yleiskäyttöinen sijoitusmakro. Makrolla voi asettaa paitsi muuttujalle uuden arvon myös monien muiden tietorakenteiden ja tallennuspaikkojen arvon. setf on myös laajennettavissa rajattomasti. Seuraavat esimerkit valottavat asiaa.

Perustilanne on lausekkeessa (setf tieto 3), jossa asetetaan muuttujan tieto arvoksi 3. Sama ajatus on yleistetty muihinkin tietorakenteisiin. Esimerkiksi funktiokutsu (elt lista 0) palauttaa listan ensimmäisen elementin. Kyseistä elementtiä voi muokata lausekkeella (setf (elt lista 0) "uusi arvo"). Vastaavalla tavalla muokataan myös taulukon tai hajautustaulun elementtejä sekä olioiden sisältämiä arvoja. Ohjelmoija voi laajentaa setf-makroa luomalla uusia sijoitusfunktioita, kuten on tehty seuraavassa esimerkissä:

```
(let ((tieto "alkuperäinen arvo"))

  (defun anna-tieto ()
    tieto)

  (defun (setf anna-tieto) (arvo)
    (setf tieto arvo)))
```

Edellä oleva ohjelma määrittelee funktion anna-tieto, joka palauttaa muuttujan tieto arvon. let-rakenteen avulla muuttuja on suljettu funktion sisään (sulkeuma), joten muuttujaan ei pääse käsiksi mistään muualta. Lisäksi ohjelma määrittelee samannimisen sijoitusfunktion setf-makroa varten, joten sen avulla tieto-muuttujaan voi vaikuttaa.

Kokeillaan ohjelmaa. Kutsutaan ensin anna-tieto-funktiota, käytetään sitten setf-makroa ja sijoitusfunktiota ja kutsutaan lopuksi funktiota uudelleen.

```
(anna-tieto) ; => "alkuperäinen arvo"
(setf (anna-tieto) "uusi arvo")
(anna-tieto) ; => "uusi arvo"
```

Tilat ja uudelleenkäynnistys

Ohjelmointikielissä on tavallisesti ominaisuus ja syntaktinen rakenne, jolla käsitellään virhetilanteita. Virheen tai muun poikkeustilanteen havainnut ohjelman osa lähettää asiasta tiedon (tilan), ja ohjelman suoritus siirtyy paikkaan, jossa poikkeustilanne käsitellään. Rakenne on kaksiosainen: tiedon lähettäjä ja poikkeustilanteen käsittelijä.

Common Lispin tilajärjestelmä on kolmeosainen, sillä mukana on myös uudelleenkäynnistys. Mihin tahansa ohjelman kohtaan voidaan määrittää uudelleenkäynnistyspisteitä. Kun tapahtuu virhe tai muu poikkeustilanne, siirrytään käsittelyosaan, mutta sieltä on mahdollista palata voimassa oleviin uudelleenkäynnistyspisteisiin.

Kolmeosaisen tilajärjestelmän avulla matalan tason funktion havaitsema virhetila voidaan lähettää ylemmän tason ohjelmalle käsiteltäväksi ja korjata jossakin muualla. Lopulta palataan matalan tason funktioon jatkamaan alkuperäistä tehtävää.

Oliot

Myös Common Lispin oliojärjestelmässä on omat erityispiirteensä. Ne voivat vaatia hieman toisenlaista ajattelutapaa, jos on tottunut muihin oliokieliin. Oliojärjestelmä muodostuu luokista, olioista, yleisfunktioista ja niiden metodeista.

> *Greenspun's Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp." – Philip Greenspun*
>
> lähde: http://philip.greenspun.com/research/

Luokilla voi olla useita rinnakkaisia kantaluokkia, ja luokat perivät ominaisuutensa kaikilta kantaluokiltaan (moniperintä). Luokkia ja niiden välisiä suhteita voi muuttaa dynaamisesti ohjelman suorituksen aikana.

Metodit eivät ole sidoksissa luokkiin, vaan ne ryhmitellään yleisfunktioiden alaisuuteen. Yleisfunktio kuvaa jotakin yleistä toimintoa, ja metodit puolestaan toteuttavat kyseisen toiminnon joidenkin luokkien näkökulmasta. Kieli tukee monimetodeja, eli metodit voidaan valita useamman argumentin perusteella. Monimetodien luonnollinen seuraus on, että metodeja ei ole järkevää sitoa yksittäiseen luokkaan. Siksi on kehitetty yleisfunktion käsite.

Rakennetaan oliomalli "kivi, sakset ja paperi" -peliin. Määritellään ensin esine-luokka ja sille alaluokiksi tarvittavat kolme esinettä. Sen jälkeen määritellään yleisfunktio ottelu, joka saa argumenteiksi kaksi esine-oliota. Yleisfunktion metodit (7 kpl) erikoistuvat erilaisiin esinevaihtoehtoihin.

```
;;; Luokat
(defclass esine () nil)

(defclass kivi (esine) nil)
(defclass sakset (esine) nil)
(defclass paperi (esine) nil)

;;; Yleisfunktio
(defgeneric ottelu (esine1 esine2))

;;; Metodit
(defmethod ottelu ((eka kivi) (toka sakset))
  (write-line "Kivi voittaa sakset."))
(defmethod ottelu ((eka kivi) (toka paperi))
  (write-line "Kivi häviää paperille."))
(defmethod ottelu ((eka sakset) (toka kivi))
  (write-line "Sakset häviää kivelle."))
(defmethod ottelu ((eka sakset) (toka paperi))
  (write-line "Sakset voittaa paperin."))
(defmethod ottelu ((eka paperi) (toka kivi))
  (write-line "Paperi voittaa kiven."))
(defmethod ottelu ((eka paperi) (toka sakset))
  (write-line "Paperi häviää saksille."))

(defmethod ottelu ((eka esine) (toka esine))
  (write-line "Tasapeli."))
```

Testataan ohjelmaa kolmella esinevaihtoehdolla:

```
(ottelu (make-instance 'kivi)
        (make-instance 'sakset))
Kivi voittaa sakset.

(ottelu (make-instance 'paperi)
        (make-instance 'sakset))
Paperi häviää saksille.

(ottelu (make-instance 'kivi)
        (make-instance 'kivi))
Tasapeli.
```

Metodeja ja niiden palauttamia arvoja on mahdollista yhdistää monilla eri tavoilla. On olemassa esimerkiksi before-, after- ja around-metodit, jotka suoritetaan ennen varsinaisia metodeja, niiden jälkeen tai niiden ympärillä. Lisätään "kivi, sakset ja paperi" -peliin yksi yhteinen around-metodi, joka kertoo ottelun alkamisesta ja päättymisestä.

```
(defmethod ottelu :around ((eka esine)
                           (toka esine))
  (write-line "Ottelu alkaa!")
  ;; Suoritetaan varsinaiset metodit:
  (call-next-method)
  (write-line "Ottelu päättyi!"))
```

Nyt yleisfunktion kutsuminen tulostaa esimerkiksi seuraavan:

```
Ottelu alkaa!
Sakset voittaa paperin.
Ottelu päättyi!
```

Määritelmästä toteutuksiin

Common Lisp on kielen määritelmä eli vain tekstiä, joka kuvaa kielen ominaisuudet. Määritelmän perusteella on kuitenkin tehty useita tietokoneohjelmia, jotka toteuttavat kielen ja tarjoavat suoritusympäristön ohjelmille. Mikään toteutuksista ei ole virallinen, sillä virallista kehittäjätahoa ei ole.

Kielen toteutukset eroavat hieman toisistaan, koska määritelmä antaa vapautta. Eroja on esimerkiksi siinä, mihin muotoon ohjelmat käännetään. Yksi toteutus kääntää ohjelmat konekielelle ja toinen puolestaan tavukoodiksi. Myös Unix-tyyppisten skriptiohjelmien kirjoittaminen voi olla mahdollista. Seuraava skriptiesimerkki hyödyntää SBCL-nimistä toteutusta:

```
#!/usr/bin/sbcl --script
(write-line "Terveisiä Lisp-ympäristöstä!")
```

Toteutukset tarjoavat myös toisistaan poikkeavia lisäominaisuuksia. Kielen määritelmä ei estä tarjoamasta enemmän kuin vaadittavat toiminnot, ja toisaalta määritelmästä puuttuukin eräitä tärkeitä nykyaikana tarvittavia ominaisuuksia. Määritelmästä puuttuu esimerkiksi verkkoliikenne, prosessin sisäiset säikeet ja rajapinta C-kielisten ohjelmakirjastojen käyttöön. Nämä kaikki sisältyvät suosituimpiin toteutuksiin.

Toteutuskohtaisten lisäominaisuuksien hyödyntäminen tarkoittaa samalla sitä, että ohjelma toimii vain kyseisessä toteutuksessa. On kuitenkin olemassa ohjelmakirjastoja, jotka piilottavat toteutuskohtaiset erot yleisen ohjelmointirajapinnan taakse. Niiden avulla voi kirjoittaa ohjelmia, jotka ovat siirrettävissä toteutuksesta toiseen.

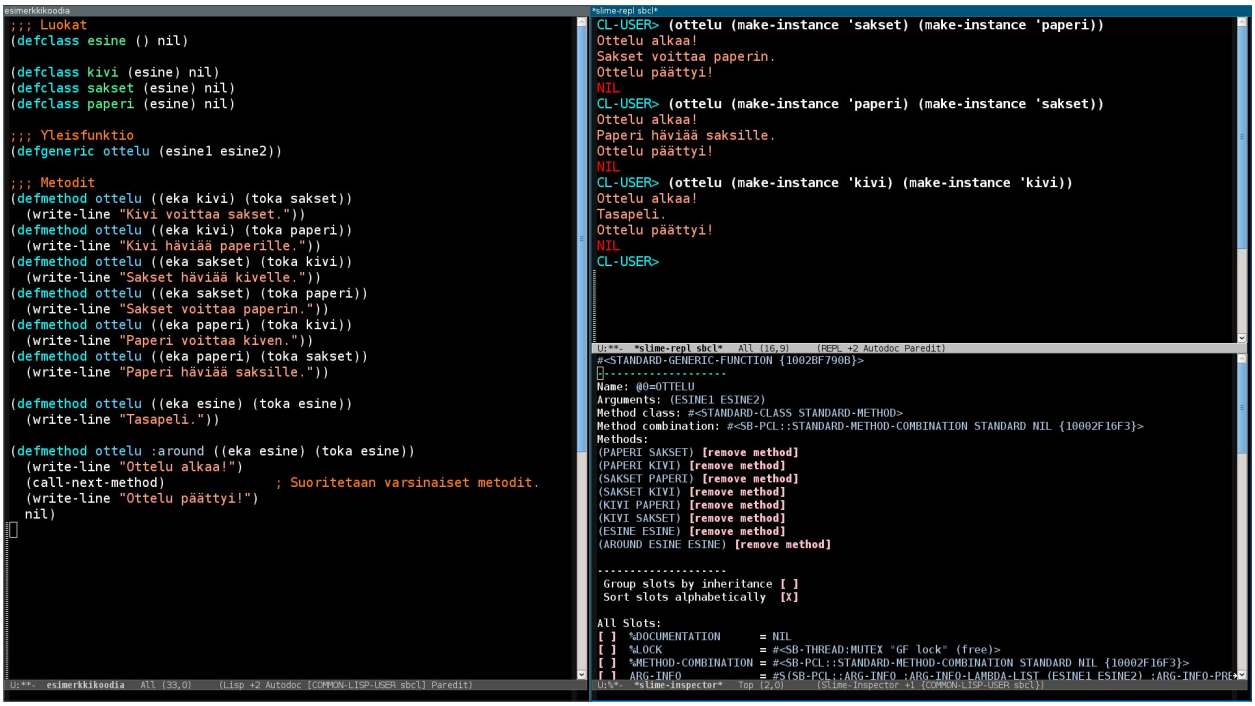
Ohjelmointi Common Lisp -kielellä alkaa siitä, että hankitaan jokin kielen toteutus ja asennetaan se tietokoneelle. On vaikeaa neuvoa, mikä toteutus kenenkin pitäisi valita, mutta aluksi lienee järkevintä turvautua suosituimpiin. Vapaista avoimen lähdekoodin periaatteella kehitettävistä toteutuksista selvästi suosituin on Steel Bank Common Lisp (lyh. SBCL). Sen jälkeen tulevat Clozure Common Lisp (CCL) ja GNU CLISP. Kaupallisia toteutuksia ovat puolestaan LispWorks ja Allegro Common Lisp (ACL).

Ensimmäiset suljesulkeiset?

Common Lispin opetteluun on olemassa hyviä oppaita mutta valitettavasti ei suomenkielisenä. Suositeltava opas on Peter Seibelin "Practical Common Lisp" (2005), joka esittelee kielen ominaisuuksia käytännönläheisesti. Kirja on saatavana painettuna, mutta se on myös vapaasti luettavissa internetistä. Paljon kehuja on saanut myös Conrad Barskin "Land of Lisp" (2010), joka opettaa kieltä hauskalla tavalla peliohjelmoinnin kautta. Hakuteokseksi ohjelmoija tarvitsee tietenkin myös itse kielen määritelmän. Se on löydettävissä internetistä nimellä "Common Lisp HyperSpec".

Mielekäs Lisp-ohjelmien kirjoittaminen vaatii muokkausohjelmalta, tekstieditorilta, hieman enemmän kuin useimmat muut ohjelmointikielet. Ohjelmat kyllä ovat tavallisia tekstitiedostoja, mutta niiden kirjoittaminen on vaivalloista, jos editori ei lainkaan ymmärrä Lisp-koodia. Oikeanlainen koodirivien sisentäminen on hyvin monitasoista, eivätkä tavalliset sarkainkohdat riitä. Siksi käytännössä vähimmäisvaatimuksena on, että editori osaa sisentää koodirivejä automaattisesti oikealla tavalla. Hyvät editorit myös tunnistavat Lispin lista- ja muut rakenteet ja sisältävät toimintoja niiden käsittelyyn.

Vapaiden ohjelmistojen joukossa kehittynein ja suosituin Lisp-kehitysympäristö on GNU Emacs -tekstieditori. Editori jo itsessäänkin on suurimmaksi osaksi toteutettu Lispin murteella, Emacs Lispillä, mutta se sopii käytet-



GNU Emacs ja Slime – vapaa kehitysympäristö Common Lispille.

täväksi muidenkin murteiden kanssa. Kun Emacsiin asentaa Slime-laajennoksen, muodostuu siitä lähes ihanteellinen Common Lispin kehitysympäristö. Koodia voi suorittaa ja testata joustavasti sitä mukaa, kun sitä syntyy. Kaupallisten Common Lisp -toteutusten mukana saattaa tulla oma editorinsa ja kehitysympäristönsä.

Lisp-yhteisö kokoontuu ircissä Freenode-verkon kanavalla #lisp ja Usenetin uutisryhmässä comp.lang.lisp. Kummassakin on läsnä lahjakkaita ohjelmoijia, jotka tuntevat Common Lispin läpikotaisin. Lisäksi on olemassa LispForum, nettiselaimella käytettävä keskustelualue, josta voi saada apua Lisp-taipaleella. Kaikkein tärkeintä on tietysti oma tutkimisen halu ja kokeileminen.

Jos tämän artikkelin kirjoittajalta kysytään, Common Lispiä ei kannata jättää pelkäksi kokeiluksi, sillä se on paljon muutakin kuin eksoottinen kuriositeetti valtakielten joukossa. Mielestäni Common Lispissä ja sen toteutuksissa yhdistyvät ihanteellisella tavalla matalatasoisten staattisten kielten suoritusteho ja korkeatasoisten dynaamisten kielten ilmaisuvoima. Voiko parempaa olla?

Kirjallisuutta

- ANSI Common Lisp -standardi eli kielen määritelmä: http://www.lispworks.com/documentation/HyperSpec/Front/index.htm
- Barski, Conrad 2010: Land of Lisp. http://landoflisp.com/
- Graham, Paul 1993: On Lisp. http://www.paulgraham.com/onlisp.html
- McCarthy, John 1979: History of Lisp. http://www-formal.stanford.edu/jmc/history/lisp/lisp.html
- Seibel, Peter 2005: Practical Common Lisp. Apress. http://www.gigamonkeys.com/book/

Suosittuja toteutuksia

- Allegro Common Lisp: http://www.franz.com/products/allegro-common-lisp/
- Clozure Common Lisp: http://ccl.clozure.com/
- GNU CLISP: http://www.clisp.org/
- LispWorks: http://www.lispworks.com/
- Steel Bank Common Lisp: http://www.sbcl.org/

Muuta hyödyllistä

- Common Lisp -projekteja: http://common-lisp.net/
- Common Lisp -wiki: http://www.cliki.net/
- Gnu Emacs -tekstieditori: http://www.gnu.org/software/emacs/
- LispForum: http://www.lispforum.com/
- Quicklisp, ohjelmakirjastojen hallintaan: http://www.quicklisp.org/

OpenGL-ohjelmointi: teksturointi ja valaistus

3D-ohjelmoinnissakin on välillä heittäydyttävä pinnalliseksi ja kiinnitettävä huomio pelkkään ulkokuoreen. Sarjan tässä osassa herätämme kappaleet uuteen loistoon.

Teksti: Mikko Rasa Kuva: Mikko Rasa, Mitol Berschewsky

Teksturointi

Todellisessa maailmassa harva pinta on täysin tasainen ja yksivärinen. Lähimmäksi pääsevät ehkä laboratoriot ja sairaalat, joiden on tarkoituskin olla steriilejä. Koska kolmiulotteisella tietokonegrafiikalla jäljitellään usein todellista maailmaa, on pintoihin saatava eloa jollain tavalla. Tapaa kutsutaan teksturoinniksi.

Yksinkertaisimmillaan tekstuuri on kolmiulotteisen kappaleen pinnalle liimattu kuva. Sen sisältö voi olla mitä tahansa, esimerkiksi sanomalehden sivu, tiiliseinä tai puunrungon kaarna. Tekstuurin kuvapisteistä käytetään termiä tekseli (engl. texel, TEXture ELement).

Monissa ihmisen tekemissä pinnoissa esiintyy toistuvia kuvioita. Tällaisia ovat vaikkapa seinäpaperi tai katukiveys. Koko seinän tai kadun tekeminen yhtenä tekstuurina vaatisi valtavan määrän tekseleitä, eikä tekstuureja mahtuisi muistiin kovin monta. Mieluummin kannattaa ottaa halutusta pinnasta vain pieni osa ja monistaa sitä. Osien täytyy kuitenkin sopia yhteen, jotta ne vierekkäin ja päällekkäin sijoitettuna muodostaisivat saumattoman pinnan.

OpenGL tarjoaa kaksi eri algoritmia toistuvien pintakuvioiden luomiseen. Oletusasetuksena on tekstuurin toistaminen sellaisenaan peräkkäin. Vaihtoehtona on peilata joka toinen toistokerta, jolloin symmetrisestä kuviosta tarvitsee pitää muistissa vain puolet. Toiston voi kytkeä myös kokonaan pois päältä. Tällä voi välttää reunoilla esiintyvät artifaktit tekstuurissa, jota ei ole tarkoitettu toistuvaksi.

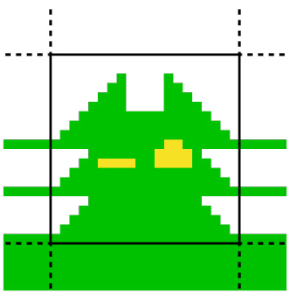
Luonnossa ei esiinny täsmällistä toistuvuutta, joten tietokonegrafiikassa vaarana on, että pinta näyttää keinotekoiselta. Tekstuuri on hyvä tehdä riittävän suureksi, ja kovin voimakkaita yksityiskohtia on syytä välttää. On myös mahdollista käyttää monitekstursointia laaja-alaisen vaihtelun lisäämiseksi.

Tekstuurin tekselit harvemmin kohdistuvat täsmälleen näytön pikseleihin. Yleensä pinta on vinossa tai tekselit ovat erikokoisia kuin pikselit. Näytteistettäessä voidaan valita lähin tekseli ja käyttää sen väriarvoa sellaisenaan. Lopputuloksena on laatikoitunut kuva tai mössöä, riippuen siitä, suurennetaanko vai pienennetäänkö tekstuuria.

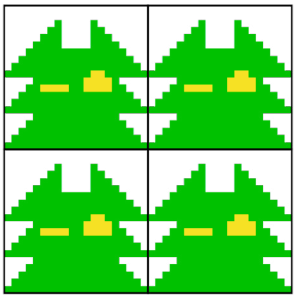
Parempi lopputulos saadaan lineaarisella suodatuksella (engl. linear filtering). Siinä valitaan tekstuurikoordinaattien ilmaisemaa pistettä ympäröivät tekselit ja interpoloidaan niiden välillä. Tässäkin tapauksessa voimakkaasti pienennetty tekstuuri puuroutuu, koska joitakin tekseleitä jätetään välistä.

Laadukkain tapa tekstuurin pienentämiseen on mipmäppäys (engl. mipmapping). Tekstuurista lasketaan etukäteen puolet pienempiä versioita aina yhden tekselin kokoon saakka. Näytteistettäessä valitaan lähimmäksi osuva suurempi ja pienempi taso, poimitaan niistä väriarvo lineaarisella suodatuksella ja lopuksi interpoloidaan lineaarisesti näiden välillä.

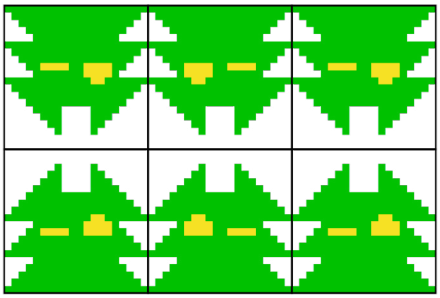
Teksturoitu pinta on paremman näköinen kuin tasavärinen, mutta sekin on



Ei toistoa: tekstuurin reunimmaiset tekselit venyvät loputtomiin.



Normaali toisto: trolliarmeija hyökkää!



Peilattu toisto: nyt niitä kasvaa jo katossakin.

Jos teksturoitua pintaa katsellaan lähes pinnan suuntaisesti, suodatuslogiikka valitsee liian pienen mipmap-tason ja seurauksena on sumentunut tekstuuri. Tähän auttaa anisotrooppinen suodatus. Se muistuttaa kaksoislineaarista suodatusta, mutta neljän tekselin muodostaman neliön sijaan käytetään jopa 16×2 tekselin suorakaidetta. Anisotrooppinen suodatus ei ole mukana vielä OpenGL 3.3:n ydinmäärittelyssä vaan löytyy laajennoksesta GL_EXT_texture_filter_anisotropic.

muodoltaan tasainen. Tämän huomaa erityisen selvästi, jos mukana on pistemäisiä valonlähteitä tai heijastuksia. Viimeisenä silauksena pinnasta pitäisi saada epätasaisen näköinen, mutta pienten yksityiskohtien tekeminen kolmioverkolla nostaisi kulmapisteiden määrää liikaa. Muhkuraisen pinnan illuusion voi luoda myös siihen tarkoitetulla tekstuurilla, niin sanotulla normaalikartalla (engl. normalmap). Sitä käytetään pinnan normaalin muunteluun kulmapisteiden välillä, jolloin valo heijastuu eri kohdista eri tavalla.

Yleisesti käytetyt kuvaformaatit pystyvät tallentamaan väriarvot ainoastaan 8-bittisinä kokonaislukuina, ja normaalikarttaan tarvitaan liukulukuja –1:n ja 1:n väliltä. Ongelma ratkaistaan skaalaamalla normaalikartan arvot 0:n ja 255:n välille tekstuuriin tallennusta varten ja muuntamalla ne shaderissa takaisin vektorimuotoon. Yleinen käytäntö on, että positiivinen Z-koordinaatti osoittaa ulospäin pinnasta, jolloin neutraali vektori saa väriarvon #8080FF. Koska normaalit usein poikkeavat neutraalista vain vähän, normaalikartoista tulee sinisävytteisiä.

Tekstuurin ei välttämättä tarvitse olla kaksiulotteinen. Yksiulotteista tekstuuria (yksittäinen rivi tekseleitä) voi käyttää esimerkiksi väriliukujen tekemiseen. Kolmiulotteisia tekstuureja käytetään tilavuudellisiin (engl. volumetric) tehosteisiin tai animoituihin pintoihin.

Eksoottisempi tekstuurityyppi on kuutiotekstuuri. Se on ikään kuin kaksi- ja kolmiulotteisen tekstuurin välimuoto. Kuutiotekstuuri koostuu kuudesta kaksiulotteisesta tekstuurista, jotka on sijoitettu kuution sivuille. Tekstuurikoordinaatteina käytetään kolmiulotteista vektoria, ja näytteistys tapahtuu kuution pinnalta vektorin osoittamasta suunnasta. Kuutiotekstuureita käytetään yleisesti ympäristökarttojen tai taivaan toteuttamiseen.

OpenGL:n tekstuurirajapinta

Tekstuurit noudattavat OpenGL:n normaalia oliorajapintaa (ks. edellinen numero). OpenGL tukee yksi-, kaksi- ja kolmiulotteisia tekstuureita sekä kuutiotekstuureja. Ne kaikki käyttävät yhteistä oliotyyppiä Texture mutta eri sidontapisteitä. Ensimmäinen sidonta määrittelee tekstuurin tyypin, eikä sitä voi muuttaa myöhemmin.

Jotta tekstuurista olisi mitään iloa, on siihen ladattava kuva. OpenGL ei tarjoa rajapintaa kuvatiedostojen lataamiseen, vaan pikselidata on saatava tiedostosta muistiin muulla tavoin. Kuvanlatauskirjastojen käyttö ei kuitenkaan kuulu tämän artikkelin aihepiiriin. Toisaalta kuvatiedostojen sijaan voidaan laskea vaikkapa fraktaaleja suoraan muistissa oleviin puskureihin.

Kun kuva on muistissa, se ladataan tekstuuriin funktiolla glTexImage<N>D, jossa <N> on ulottuvuuksien määrä. Samalla määritellään tekstuurin koko ja väriformaatti. Tekstuurin sisäinen säilytysformaatti ja syötettävän datan formaatti annetaan erikseen. Tämä säästää vaivaa eksoottisempien formaattien kanssa, koska ohjelman ei tarvitse käsitellä esimerkiksi 11-bittisiä liukulukuja.

Level-parametria käytetään mipmap-tasojen lataamiseen. Täysikokoinen kuva on taso 0, ja jokainen sitä korkeampi taso on puolet edellistä pienempi. Puolikkaat tekselit pyöristetään alaspäin. Viimeisellä tasolla tekstuurin koko kaikissa suunnissa on yksi tekseli. Jos jonkin tason koko on väärä, tekstuuria ei voi käyttää.

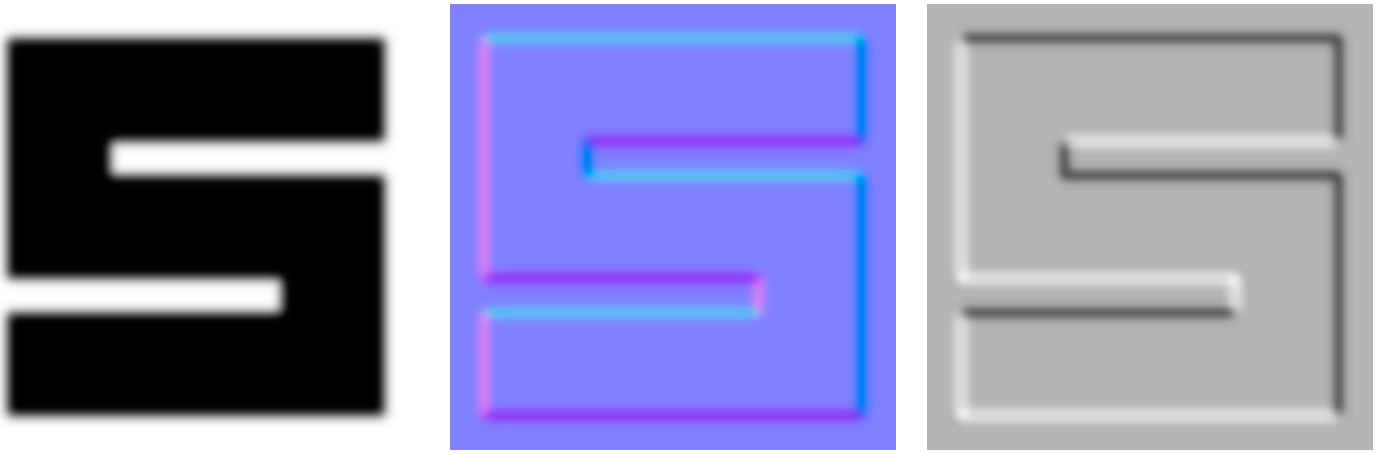
Jos data-osoitin on null, tekstuurille varataan tilaa, mutta sen sisältö on määrittelemätön. Näin voidaan luoda esimerkiksi kuvapuskurina käytettävä tekstuuri eikä tarvitse turhaan varata suurta muistialuetta, joka heti perään vapautettaisiin.

Kuutiotekstuurille ei ole erillistä funktiota, vaan kukin sivu ladataan erikseen kaksiulotteisena tekstuurina. Sidontapisteenä käytetään tällöin yhtä kuudesta vakiosta, jotka kuvaavat kuution sivuja.

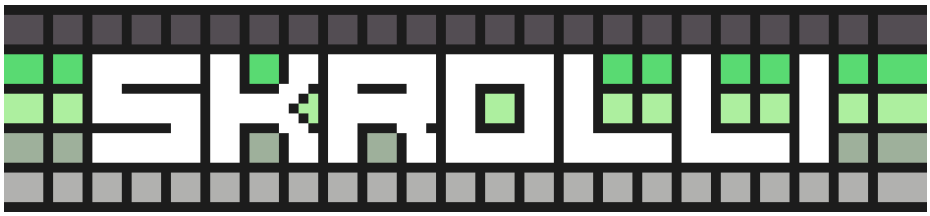
Tekstuurin suodatus-, toisto- ja muut asetukset asetetaan funktioilla glTexParameteri ja glTexParameterf. Erona näillä on asetettavan parametrin tyyppi. Suodatusparametreihin on kiinnitettävä erityistä huomiota, koska GL_MIN_FILTER-parametrin oletusarvo on GL_NEAREST_MIPMAP_LINEAR eikä mipmapeja käyttävä tekstuuri näy, ellei kaikkia mipmap-tasoja ole määritelty.

Monitekstuurointia varten on olemassa funktio glActiveTexture, joka asettaa käytettävän teksturointiyksikön. Parametrina annetaan tekstuuriyksikköä kuvaava symbolinen vakio. On tärkeää huomioida, että kaikki tekstuurioperaatiot kohdistuvat aktiiviseen tekstuuriyksikköön. Siinä piilee sudenkuoppa, sillä yksinkertaista teksturointia käyttävä koodi saattaa toimia väärin, jos moniteksturoiva osuus jättää aktiiviseksi muun kuin ensimmäisen yksikön.

Tekstuuriolion rakentaminen on vasta puoli voittoa. Jotta kuva näkyisi kappaleen pinnalla, on shaderi saatava käyttämään sitä. Koska tekstuureja voi olla käytössä useita samanaikaisesti, on

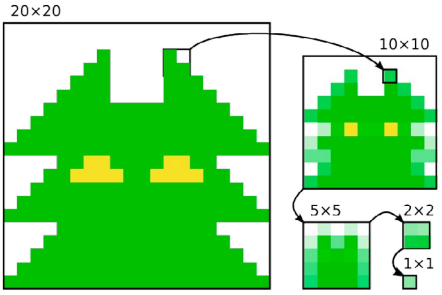


Korkeuskartta, siitä tehty normaalikartta ja valaistu lopputulos.



NEAREST-suodatuksella pikselit suurenevat terävärajaisiksi neliöiksi.

LINEAR-suodatus toimii hyvin valokuvamaiselle materiaalille, mutta tässä tapauksessa siitä on enemmän haittaa kuin hyötyä.



Mipmapeilla voidaan hallita pienennetyn kuvan laatua. Jokaisen tason voi halutessaan tehdä erikseen.

Ennen versiota 3.0 OpenGL:n kiinteässä toiminnallisuudessa oli tarjolla valaistusmalli. Se perustui verteksikohtaiseen valaistukseen ja oli siten varsin karkea, erityisesti valonlähteiden lähellä. Tämäkin ominaisuus jätettiin pois muun kiinteän laskennan mukana.

shaderille välitettävä tieto siitä, mitä tekstuuria tai tekstuureja halutaan käyttää. Se onnistuu käyttämällä erityisiä näytteistäjätyyppejä, joita on kullekin tekstuurityypille omansa. Esimerkiksi kaksiulotteisen tekstuurin näytteistäjä on sampler2D. Näytteistäjien on oltava uniform-muuttujia. Isäntäohjelman puolelta niihin asetetaan kokonaislukuarvo, joka kertoo käytettävän teksturointiyksikön järjestysnumeron. Shaderin puolella näytteistäjiä voi käyttää ainoastaan näytteistysfunktion parametrina.

Tekstuuri on myös sijoitettava oikein suhteessa kappaleen muotoihin. Yleensä kappaleen kulmapisteille määritellään etukäteen tekstuurikoordinaatit ja kulmapisteshaderi vain välittää saamansa syötteen eteenpäin pikselishaderille. On kuitenkin myös tilanteita, joissa kulmapisteshaderin täytyy muunnella tekstuurikoordinaattia tai jopa laskea se muiden syötteiden perusteella.

Jokaiselle näytteistäjätyypille on vastaava näytteistysfunktio. Esimerkiksi kaksiulotteiselle tekstuurille se on texture2D. Näytteistysfunktion ensimmäinen parametri on näytteistettävä tekstuuri ja toinen parametri tekstuurikoordinaattivektori. Kaikki näytteistysfunktiot palauttavat nelikomponenttisen vektorin, joka sisältää RGBA-arvot. Komponentit ovat normaalisti välillä nollasta yhteen, mutta joillakin väriformaateilla on eri arvoalue. Mikäli tekstuurin määrittelyssä on puutteita, palautetaan vec4(0, 0, 0, 1).

Valaistus

Toinen realistisen grafiikan kulmakivi on valaistus. Ilman sitä teksturoidutkin kappaleet näyttävät latteilta ja muotoja on vaikeaa erottaa. Valaistuksen toteuttaminen on onneksi varsin yksinkertaista.

Valonlähteitä on kahta eri päätyyppiä. Suunnattu valo (engl. directional light) on niistä yksinkertaisempi. Se osuu kaikkialle samassa suunnassa, joten laskutoimitukset helpottuvat huomattavasti. Yleisin esimerkki suunnatusta valosta on aurinko.

Toinen päätyyppi on pistemäinen valo (engl. omnidirectional light, omni light), jolla on sijainti mutta ei suuntaa. Valaistuksen muodostamiseksi on kullekin pisteelle laskettava erikseen valon tulosuunta. Esimerkiksi monet katulyhdyt ovat likimain pistemäisiä valonlähteitä.

Jos pistemäinen ja suunnattu valo yhdistetään, saadaan spottivalo (engl. spot light). Sillä on sekä sijainti että suunta ja lisäksi myös valokeilan leveys. Hyvä esimerkki spottivalosta on taskulamppu.

Valaistuslaskennan peruspalikka on vektorien pistetulo. Valaistuksen kirkkaus riippuu valon tulosuunnan ja pinnan normaalin välisen kulman kosinista, joka on yksikkövektoreilla sama kuin pistetulo. Yleensä käytetään pinnasta valonlähteeseen osoittavaa vektoria. Pistetulo on positiivinen, kun pinta on valonlähdettä kohti. Negatiiviset tulokset muutetaan nollaksi.

Kuten monessa muussakin asiassa, myös valaistuksessa voidaan tehdä valinta suorituskyvyn ja ulkonäön välillä. Valaistuksen voimakkuuden laskeminen kulmapisteshaderissa ja interpolointi pinnan yli tuottaa vain karkean likiarvon mutta on erittäin nopeaa. Tarkasti todellisuutta jäljittelevä valaistus vaatii paljon pikselikohtaista laskentaa. Sitä tarvitaan silloin, kun käytetään normaalikarttoja ja eräitä muita tehosteita.

Vektorien interpolointi tapahtuu komponenteittain lineaarisesti. Mikäli interpoloinnin päätepisteet poikkeavat toisistaan enemmän kuin aivan vähän, välivaiheet eivät pysy yksikköpituisina. Tämä aiheuttaa tummemman valaistuksen pintojen keskelle ja kirkkaamman kulmapisteiden läheisyyteen. Vektorit on siksi muistettava normalisoida ennen pistetulon ottamista.

Valaistusta laskettaessa on kiinnitettävä huomiota myös siihen, että laskutoimituksiin osallistuvat vektorit ovat samassa koordinaatistossa. Tasaisten pintojen valaistukseen käytetään tavallisesti kameran koordinaatistoa. Koska esimerkiksi pelimaailman osien sijainnit on käytännöllisintä ilmaista maailman koordinaatistossa, on grafiikkamoottorin huolehdittava koordinaatistomuunnoksesta, kun se syöttää valojen ominaisuuksia shadereille.

Normaalikartat ovat hankalampi tapaus. Normaalit tallennetaan karttaan suhteessa kolmioverkon määräämiin pinnanmuotoihin, ja niiden muuntaminen kameran koordinaatistoon vaatisi paljon laskentaa pikselishaderissa. Helpompaa onkin laskea valonlähteen suunta pinnan koordinaatistossa. Tämä voidaan tehdä jo kulmapisteshaderissa ja siten välttää kallista pikselikohtaista laskentaa. Kulmapisteille on sitä varten lisättävä kaksi tangenttia, jotka kertovat normaalikartan orientaation pinnalla.

Yksittäisten valolähteiden lisäksi lähes jokaisessa ympäristössä on myös hajavaloa (engl. ambient light), jota tulee kaikista suunnista. Se esitetään kaikkien pisteiden valaistukseen lisättävänä vakioarvona. Ulkotiloissa voi parempaan lopputulokseen päästä käyttämällä puolipallovaloa (engl. hemispherical light, hemi light), joka on ikään kuin suunnatun valon ja hajavalon välimuoto. Valonlähteen ja normaalin välinen pistetulo lasketaan kuten suunnatulle valolle, mutta tulos skaalataan niin, että vasta täysin valonlähteestä poispäin osoittava pinta on täysin pimennossa.

```
struct Vertex
{
  float x, y, z;
  float nx, ny, nz;
  float u, v;
  float tx, ty, tz;
  float bx, by, bz;
};
```

Listaus 1: Kulmapistestruktuuri, jossa ovat teksturointiin tarvittavat muuttujat.

Esimerkkiohjelma

Esimerkkiohjelmassa on tällä kertaa runsaasti uusia asioita, joten käymme tässä läpi vain pääkohdat.

OpenGL:n olioiden hallinta käsipelillä käy äkkiä työlääksi. Hätiin tulevat C++:n luokat ja RAII-periaate. Kun OpenGL-olio luodaan luokan konstruktorissa ja vapautetaan destruktorissa, voidaan niihin soveltaa normaalia C++:n muistinhallintaa.

Object-luokka kuvaa kolmiulotteista esinettä ja pitää sisällään sen tarvitsemat puskurit. Rakentajalle annetaan parametrina taulukot vertekseistä ja indekseistä. Piirtokomentojen vähentämiseksi ja rajapintojen yksinkertaistamiseksi koko esine piirretään yhtenä kolmioketjuna.

Kulmapistestruktuuri on saanut uusia jäseniä teksturointia varten. U ja v ovat tekstuurikoordinaatit, ja t-alkuisiin muuttujiin tallennetaan pinnan tangentti ja b-alkuisiin binormaali. Listauksessa 1 on uusi struktuuri kokonaisuudessaan.

```
int Shader::get_uniform_location(const string &name)
{
  map<string, int>::iterator i = uniforms.find(name);
  if(i!=uniforms.end())
    return i->second;

  int loc = glGetUniformLocation(program_id, name.c_str());
  uniforms[name] = loc;

  return loc;
}

void Shader::set_uniform(const string &name, float x, float y, float z)
{
  int loc = get_uniform_location(name);
  if(loc>=0)
    glUniform3f(loc, x, y, z);
}
```

Listaus 2: Uniform-muuttujien käsittelyfunktiot.

Esineiden luominen on myös siirretty omiin funktioihinsa, jotka ottavat parametreina luotavan esineen mitat. Koska useimpia esineitä ei ole mahdollista esittää yhtenä kolmioketjuna, käytämme primitiivien uudelleenkäynnistystä (engl. primitive restart). Sen avulla tietty indeksi voidaan määritellä tarkoittamaan uuden samantyyppisen primitiivin aloittamista.

Shader-luokka huolehtii shaderien kääntämisestä ja uniform-muuttujien asettamisesta. Rakentajalle annetaan verteksi- ja pikselishaderien lähdekoodit merkkijonoina. Jos käännöksessä tapahtuu virhe, rakentaja heittää poikkeuksen.

Listauksessa 2 on uniform-muuttujien käsittelyyn käytettävät funktiot. Muuttujien arvoja joudutaan muuttamaan usein, joten GL-kutsujen vähentämiseksi säilytämme niiden sijainnit paikallisesti.

Jokaiselle eri uniform-tyypille tarvitaan oma asetusfunktionsa, joista on tilan säästämiseksi listattu tässä vain yksi. Ylikuormituksen ansiosta ne voivat olla samannimisiä. Uniform-muuttujan asetus kohdistuu aina aktiiviseen shaderiin, joten shaderi on muistettava ottaa käyttöön ensin.

Texture-luokka on varsin yksinkertainen. Rakentajalle annetaan parametrina

Lunttilappu OpenGL:n tekstuurirajapintaan

Funktiot

void **glGenTextures**(sizei n, uint *textures);
void **glBindTexture**(enum target, uint texture);
void **glDeleteTextures**(sizei n, const uint *textures);
void **glTexImage3D**(enum target, int level, int internalformat, sizei width, sizei height, sizei depth, int border, enum format, enum type, const void *data);
void **glTexImage2D**(enum target, int level, int internalformat, sizei width, sizei height, int border, enum format, enum type, const void *data);
void **glTexImage1D**(enum target, int level, int internalformat, sizei width, int border, enum format, enum type, const void *data);
void **TexParameteri**(enum target, enum pname, int param);
void **TexParameterf**(enum target, enum pname, float param);
void **glActiveTexture**(enum texture);

Tärkeimmät parametrit

GL_MIN_FILTER
suodatus pienennettäessä (minification)
GL_MAG_FILTER
suodatus suurennettaessa (magnification)
GL_WRAP_S
ensimmäisen ulottuvuuden toisto
GL_WRAP_T
toisen ulottuvuuden toisto
GL_WRAP_R
kolmannen ulottuvuuden toisto

Suodattimet

GL_NEAREST
ei suodatusta
GL_LINEAR
lineaarinen suodatus
GL_LINEAR_MIPMAP_LINEAR
lineaarinen suodatus mipmapeilla

Toistotavat

GL_CLAMP_TO_EDGE
ei toistoa, reunimmaiset tekselit jatkuvat äärettömyyteen
GL_REPEAT
normaali toisto
GL_MIRRORED_REPEAT
peilattu toisto

Sidontapisteet

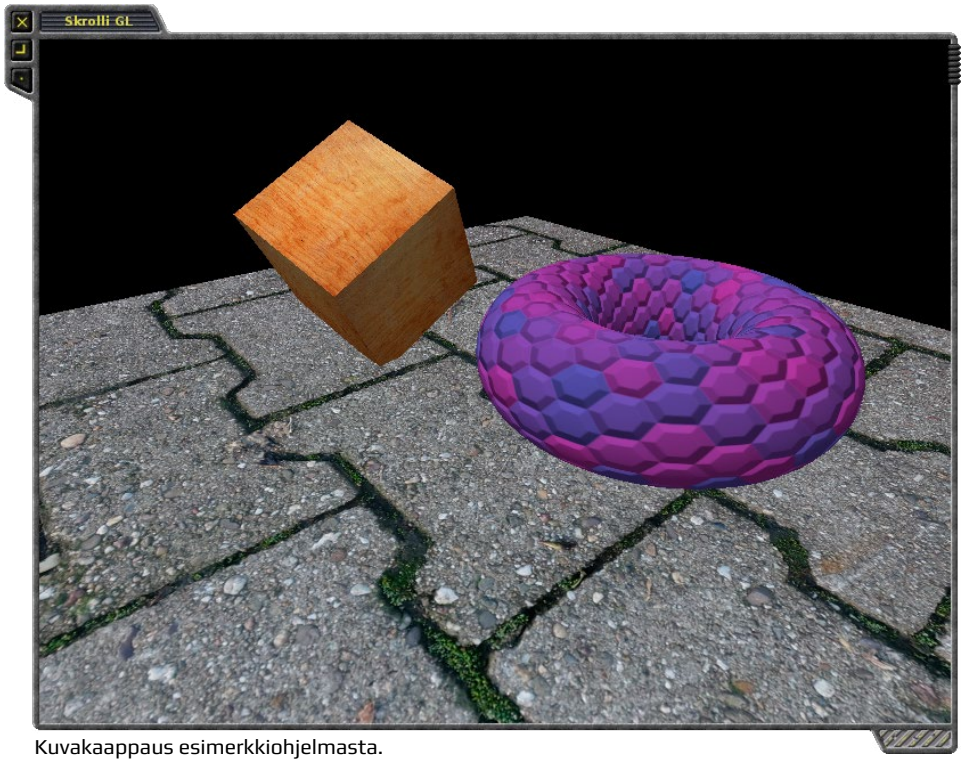
GL_TEXTURE_1D
yksiulotteinen tekstuuri
GL_TEXTURE_2D
kaksiulotteinen tekstuuri
GL_TEXTURE_3D
kolmiulotteinen tekstuuri
GL_TEXTURE_CUBE_MAP
kuutiotekstuuri

Väriformaatit

GL_RED
yksi värikanava, ainoastaan R-komponentti käytössä
GL_RG
kaksi värikanavaa, R- ja G-komponentit käytössä
GL_RGB
kolme värikanavaa
GL_RGBA
neljä värikanavaa

Yllä olevat formaatit eivät määrittele bittisyvyyttä, vaan toteutus valitsee sopivan. Bittisyvyyden voi määritellä lisäämällä sen vakion perään:

GL_fmt8
8 bittiä per komponentti
GL_fmt16
16 bittiä per komponentti
GL_fmt16F
16 bittiä per komponentti, liukuluku
GL_fmt32F
32 bittiä per komponentti, liukuluku



Kuvakaappaus esimerkkiohjelmasta.

```
void main()
{
  gl_Position = projection*modelview*in_position;
  mat3 normal_matrix = mat3(modelview);
  mat3 tbn_matrix = normal_matrix*mat3(in_tangent, in_binormal, in_normal);
  texcoord = in_texcoord;
  tbn_light = light_direction*tbn_matrix;
}
```

Listaus 3: Normaalikarttaa käyttävä kulmapisteshaderi.

```
void main()
{
  vec4 color = texture2D(texture, texcoord);
  vec3 normal = texture2D(normalmap, texcoord).rgb*2.0-1.0;
  float intensity = max(dot(normalize(tbn_light), normal), 0.0);
  out_color = vec4(color.rgb*(0.4+intensity*0.6), color.a);
}
```

Listaus 4: Normaalikarttaa käyttävä pikselishaderi.

ladattavan kuvatiedoston nimi. Sen lisäksi luokassa on ainoastaan sidontafunktio.

Myös vektorit ja matriisit ansaitsevat omat apuluokkansa. Pärjäämme varsin maltillisella määrällä operaatioita. Matriisiluokka sisältää myös joukon staattisia funktioita perusmuunnosten luomiseen. Kertolaskuoperaattorilla ketjuttamalla niistä voi kasata monimutkaisempia muunnoksia.

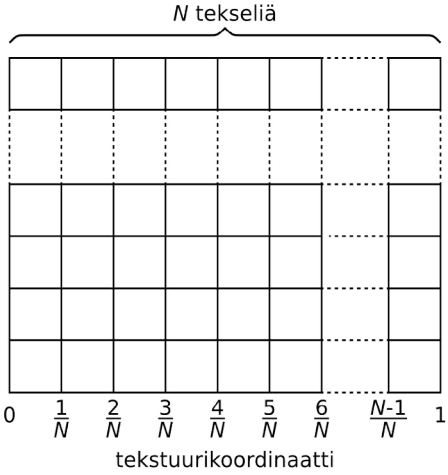
Listauksissa 3 ja 4 on normaalikarttaa käyttävän kulmapiste- ja pikselishaderin lähdekoodi. Tilan säästämiseksi muuttujien esittelyt on jätetty pois. Valon suunnan muunnoksessa TBN-avaruuden matriisi on poikkeuksellisesti oikealla puolella, koska kyseessä on kohdeavaruuden matriisi lähdeavaruudessa esitettynä.

Pinnan normaalia ei välitetä shaderien välisessä rajapinnassa, koska TBN-avaruudessa neutraali normaali on vakio. Sen sijaan pikselishaderi lukee pikselikohtaisen normaalin tekstuurista ja skaalaa rgb-arvot takaisin oikealle arvoalueelle. Max-funktio estää valaistusarvon muuttumisen negatiiviseksi, jos pinta osoittaa poispäin valonlähteestä. Voimakkuuteen lisätään vielä vakioarvo, jotteivät varjon puolella olevat pinnat olisi täysin mustia.

Tehtäviä ja haasteita:

- Koeta tehdä lisää erilaisia esineitä.
- Ota valokuvia tapetistasi, talosi seinästä tai pihatiestä ja tee niistä tekstuureja.
- Muuntele kameran reittiä esineiden ympärillä ja laita esineet liikkumaan.
- Esineiden luominen ohjelmakoodilla on pidemmän päälle hankalaa. Etsi jonkin 3D-tiedostomuodon kuvaus ja yritä kirjoittaa sille latausfunktio, tai keksi oma.
- Joitakin esimerkkiohjelman mukana tulevia tekstuureja ei voi toistaa saumattomasti. Osaatko tehdä niistä (tai jostain muusta kuvasta) saumattomia?
- Lue seuraavalla sivulla oleva osio valaistuksen teoriasta ja koeta lisätä shadereihin peiliheijastukset.
- Esimerkkishaderin valonlähde on suunnattu. Yritä tehdä siitä pistemäinen tai kohdevalo.

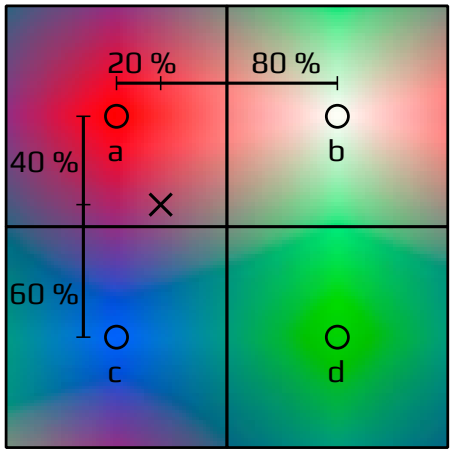
Teorialisä: tekstuurien näytteistys



Tekstuuri koostuu suorakulmaisesta taulukosta tekseleitä, joihin viitataan tekstuurikoordinaateilla. OpenGL käyttää normalisoituja tekstuurikoordinaatteja, eli koordinaatit ovat aina nollan ja yhden välillä. Jos tekstuurin koko on N tekseliä, yhden tekselin koko tekstuurikoordinaateissa ilmaistuna on 1/N.

Näytteistettävää tekseliä valittaessa tekselin sijainti määräytyy sen keskipisteen mukaan. Ilman suodatusta valitaan yksinkertaisesti lähin tekseli. Lineaarisessa suodatuksessa käytetään kullakin akselilla sekä edellistä että seuraavaa tekseliä ja lasketaan interpolaatioarvo sen mukaan, missä kohtaa tekselien välillä ollaan.

Ennen näytteistystä tekstuurikoordinaatista vähennetään lähin pienempi kokonaisluku. Sen seurauksena tekstuuri toistuu loputtomiin. Tämä vaikuttaa myös lähimpien tekselien valintaan li-



○ tekselin keskipiste
× näytteistyspiste
x = (0.8·0.6)a + (0.2·0.6)b
+ (0.8·0.4)c + (0.2·0.4)c

Lineaarinen suodatus kaksiulotteisessa tekstuurissa.

neaarisessa interpolaatiossa, jos koordinaatti on lähellä tekstuurin reunaa. Mikäli vastakkaisessa reunassa oleva tekseli on erivärinen, syntyy reunaan näkyviä artifakteja.

Toistoparametrilla voidaan vaikuttaa tekstuurikoordinaattiin ennen näytteistystä. Normaali toisto päästää koordinaatin läpi sellaisenaan. Peilattu toisto toimii muuten samoin, mutta parittomilla kokonaisluvuilla koordinaatti käännetään ympäri vähentämällä se yhdestä. Toistuminen voidaan estää rajoittamalla koordinaatin arvo välille 0.5/N – (N–0.5)/N. Oudolta vaikuttavat rajat estävät aiemmin mainittujen artifaktien syntymisen.

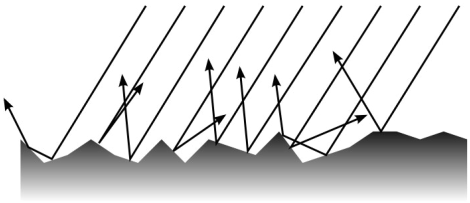
Teorialisä: valaistusmallit

Todellisessa maailmassa valo koostuu fotoneista, joita valonlähde tuottaa tietyn määrän aikayksikköä kohti. Ne heijastuvat pinnasta toiseen, kunnes lopulta päätyvät silmään ja tuottavat valoaistimuksen. Suurempi määrä fotoneita aistitaan kirkkaampana valona. Tummat pinnat imevät osan fotoneista.

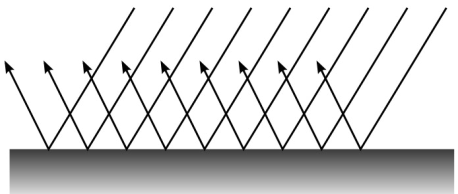
Heijastuneen fotonin suunta riippuu pinnan mikroskooppisesta rakenteesta. Tasaiseltakin vaikuttava pinta voi atomien tasolla olla rosoinen, jolloin fotonit heijastuvat satunnaisiin suuntiin. Tätä kutsutaan diffuusiksi heijastukseksi (engl. diffuse reflection). Esimerkistä käy käsittelemätön puupinta tai puuvillakangas.

Erittäin tasaisesta pinnasta, kuten vaikkapa kiillotetusta metallista tai peililasista, lähes kaikki fotonit heijastuvat samaan suuntaan. Näin syntyy spekulaarinen heijastus (engl. specular reflection).

Monissa materiaaleissa heijastuskuvio on jotain tältä väliltä. Himmeä metallipinta ja lakattu puu heijastavat valonlähteet kirkkaina pisteinä tai läiskinä, mutta peilikuvaansa niistä ei näe. Joissakin tapauksissa heijastuskuvio riippuu valon tulosuunnasta ja materiaalin sanotaan olevan anisotrooppinen. Harjattu metalli on tällainen materiaali. On jopa mahdollista luoda aivan muunlaisia pintarakenteita, kuten valon takaisin tulosuuntaansa heijastava heijastin.



Hajaheijastus: valonsäteet heijastuvat satunnaisiin suuntiin.



Peiliheijastus: valonsäteet heijastuvat samaan suuntaan.

Jos pinta on vinossa valonlähteen suuntaan nähden, sama valomäärä leviää laajemmalle alueelle. Kääntäen pintaan osuva valomäärä pinta-alaa kohden ja siten pinnan aistittu kirkkaus vähenee. Jos fotonikimpun leveys on l ja valaistu alue a, valaistuksen suhteellinen voimakkuus on l/a. Täydentämällä kuvio kolmioksi ja merkkaamalla valon suunnan ja pinnan normaalin välistä kulmaa α:lla nähdään, että l/a = cosα, joka saadaan näppärästi laskettua vektorien pistetulona.

Reaaliaikaisessa kolmiulotteisessa grafiikassa on viime kädessä kyse huijauksesta. Tehokkaimmankaan näytönohjaimen laskentateho ei riitä sellaisiin monimutkaisiin integraaleihin, joita todellisuuden pikkutarkka simulointi vaatii, ainakaan jos samalla halutaan myös sulavaa liikettä. Vuosikymmenten varrella on kehitetty useita valaistusmalleja, jotka tuottavat kelvollisen arvion valaistuksesta olematta liian vaativia.

Valon suunta pintaan nähden on harvoin vakio. Pinta voi olla kaareva, valonlähde pistemäinen tai molempia. Ensimmäiset valaistusmallit laskivat kappaleen jokaiselle tasopinnalle yhden valaistusarvon, jolloin lopputulos oli varsin kulmikkaan näköinen. Henri Gouraud keksi laskea valaistusarvot verteksikohtaisesti ja interpoloida niitä pintojen yli. Näin saadaan piilotettua tasopintojen reunat, mutta näkyviä artifakteja esiintyy silti monissa tilanteissa.

Bui Tuong Phongin kehittämässä, hänen mukaansa nimetyssä Phong-mallissa valaistus jaetaan erilliseen diffuusiin ja spekulaariseen komponenttiin. Diffuusin valaistuksen voimakkuus lasketaan pinnan normaalin ja valon suunnan pistetulona. Spekulaarista komponenttia varten lasketaan ensin heijastuvan valon suunta ja muodostetaan pistetulo sen ja katselusuunnan välillä. Saatu arvo korotetaan potenssiin suuntautuneen heijastuskuvion simuloimiseksi. Suurempi eksponentti saa aikaan pienemmän heijastuksen ja vaikutelman kiiltävämmästä materiaalista.

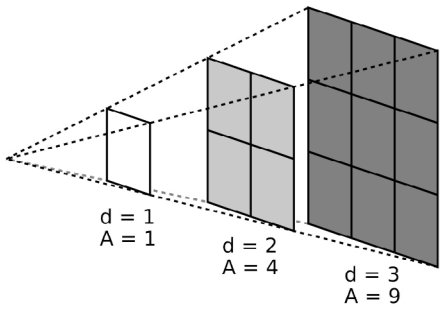
Phong-mallista käytetään usein Jim Blinnin kehittämää muunnelmaa, jossa spekulaaristen heijastusten laskentaan käytetään valon heijastusvektorin sijasta valonlähteen ja katselupisteen puoliväliin osoittavaa vektoria. Blinn–Phong-malli on tehokkaampi suunnattuja valonlähteitä käytettäessä, koska samaa puolivälivektoria voidaan käyttää koko kuvalle.

Niin ikään Phongin kehittämä on parempi interpolaatioalgoritmi, jossa valaistusarvojen sijaan interpoloidaan pinnan normaalia. Tämä vaatii enemmän laskentatehoa, koska interpoloitavia muuttujia on yhden sijaan kolme ja vektori pitää vielä normalisoida kunkin pikselin valaistusta laskettaessa. Algoritmi kuitenkin pystyy laskemaan spekulaariset heijastukset myös laajojen pintojen keskellä ja mahdollistaa monia kehittyneempiä valaistustekniikoita.

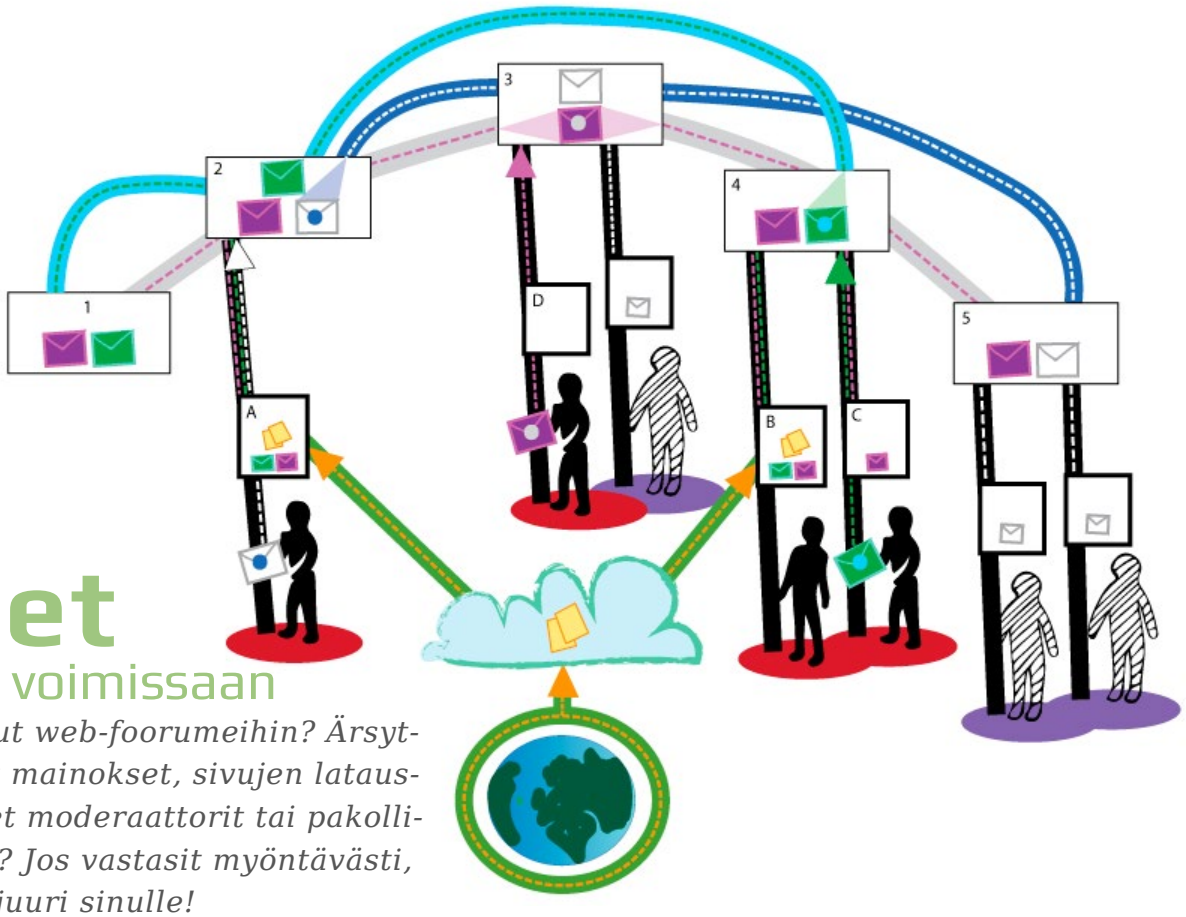
Kohtauskulman lisäksi valaistuksen voimakkuuteen vaikuttaa etäisyys valonlähteestä. Suunnatut valonlähteet, kuten aurinko, ovat niin kaukana, että etäisyys on käytännössä vakio eikä havaittavaa heikkenemistä tyypillisen pelimaailman alueella tapahdu. Pistemäiset valonlähteet kuitenkin sijaitsevat pelimaailmassa muiden esineiden keskellä, jolloin etäisyyden vaikutus täytyy ottaa huomioon.

Tässäkin on kyse saman valomäärän leviämisestä laajemmalle alueelle. Mekanismin hahmottamista helpottaa, jos tarkastellaan valonlähteestä tiettynä hetkenä lähteneitä fotoneja yhtenäisenä pallon muotoisena rintamana. Säteen ja pinta-alan välillä on neliöllinen suhde, eli säteen kaksinkertaistuessa pinta-ala nelinkertaistuu. Valomäärä pinta-alaa kohden pienenee vastaavasti, ja sen sanotaan olevan kääntäen verrannollinen etäisyyden neliöön, kaavana ilmaistuna $E_v \propto 1/r^2$.

Jakolaskulla on ikävä taipumus tuottaa hyvin suuria tuloksia, kun jakaja lähestyy nollaa. Tietokonemonitorin dynamiikka on huonompi kuin silmän, ja lähellä valonlähdettä olevien kappaleiden kirkkaus saturoituu helposti, ellei asialle tehdä jotain. Helppo ratkaisu on poiketa jälleen tosimaailman fysiikasta ja lisätä jakoviivan alle vakiotermi, joka pitää jakajan positiivisena. Kaukana valonlähteestä eroa ei juuri huomaa.



Valon voimakkuus heikkenee etäisyyden kasvaessa.



Usenet

Vanhus yhä voimissaan

Oletko tuskastunut web-foorumeihin? Ärsyttävätkö välkkyvät mainokset, sivujen latausajat, mielivaltaiset moderaattorit tai pakolliset rekisteröinnit? Jos vastasit myöntävästi, Usenet voi sopia juuri sinulle!

Teksti: Lauri Alanko Kuvat: Lauri Alanko, Mitol Berschewsky

Kun Linus Torvalds oli aloittanut uuden käyttöjärjestelmänsä kehittämisen, hän kirjoitti asiasta lyhyen viestin, jonka lähetti Helsingin yliopiston palvelimelle Minix-käyttöjärjestelmän keskusteluryhmään. Jos innokkaimmat opiskelutoverit seurasivat samaa ryhmää juuri tuolloin, he saattoivat nähdä viestin ensimmäisinä maailmassa. Muutaman hetken päästä yliopiston palvelin lähetti viestin koneeseen Espoossa, josta se välitettiin Amsterdamiin ja sieltä yhä eteenpäin. Muutaman tunnin sisällä suomalaisen käyttöjärjestelmän syntysanat oli kopioitu ympäri maailmaa, tuhansille koneille, paikallisten lukijoiden nähtäville.

Vuosi oli 1991, ja järjestelmä oli Usenet, tuolloin internetin suosituin tapa käydä julkista keskustelua. Nykyajan sosiaalista mediaa ei ollut, sillä koko WWW oli vasta lapsenkengissään. Vain paria viikkoa aiemmin Tim Berners-Lee oli kertonut omasta projektistaan julkisesti ”alt.hypertext”-ryhmässä, Usenetissä sekin.

Internetin alkuperäinen keskustelujärjestelmä porskuttaa yhä luotettavasti, vailla krumeluureja, kaukana julkisuuden valokeiloista. Usenetillä on pitkä ja maineikas historia. Se on yksi internetin vanhimmista viestintäpalveluista, heti sähköpostin jälkeen. Nykyään sitä verhoaa salaperäisyys: keskusteluihin ei törmää tavallisilla www-hauilla kuin satunnaisesti. Paljon helpompaa on löytää sivustoja, jotka mainostavat Usenetiä eliittipiirien salaisena piratismiverkkona. On siis syytä valottaa, mistä on kyse: avoimesta, hajautetusta viestintävälineestä, jolla on paljon tarjottavaa.

Vertaisverkkojen esi-isä

Usenet syntyi 1970-luvun lopulla Yhdysvaltain Pohjois-Carolinassa. Kahden yliopiston sisäiset tiedotusjärjestelmät alkoivat vaihtaa viestejä keskenään puhelinlinjoja pitkin, ja käyttäjät pääsivät lukemaan toistensa kirjoituksia. Idea osoittautui suosituksi, ja kun tekniikka oli kerran kehitetty, monet muut korkeakoulut ja muutamat teknologiafirmat osallistuivat viestien vaihtoon. Kun koneiden käyttäjät pääsivät näin yhteyteen keskenään, syntyneen verkon nimeksi tuli ”Usenet”, käyttäjien oma verkko.

Alkuaikojen keskustelut Usenetissä koskivat lähinnä ohjelmistoja ja tietoverkkoja. Pian aiheet levisivät harrastuksiin, tieteeseen ja politiikkaan. Koska järjestelmää käytettiin alun perin tiedottamiseen, eri aiheiden keskusteluryhmiä kutsutaan yhä nimellä newsgroups, ”uutisryhmät”. Samasta syystä viestejä kutsutaan usein ”artikkeleiksi” ja palvelimia ”uutispalvelimiksi”. Termit ovat hämääviä, ja suomeksi on selkeintä puhua tuttavallisesti ”nyysseistä”.

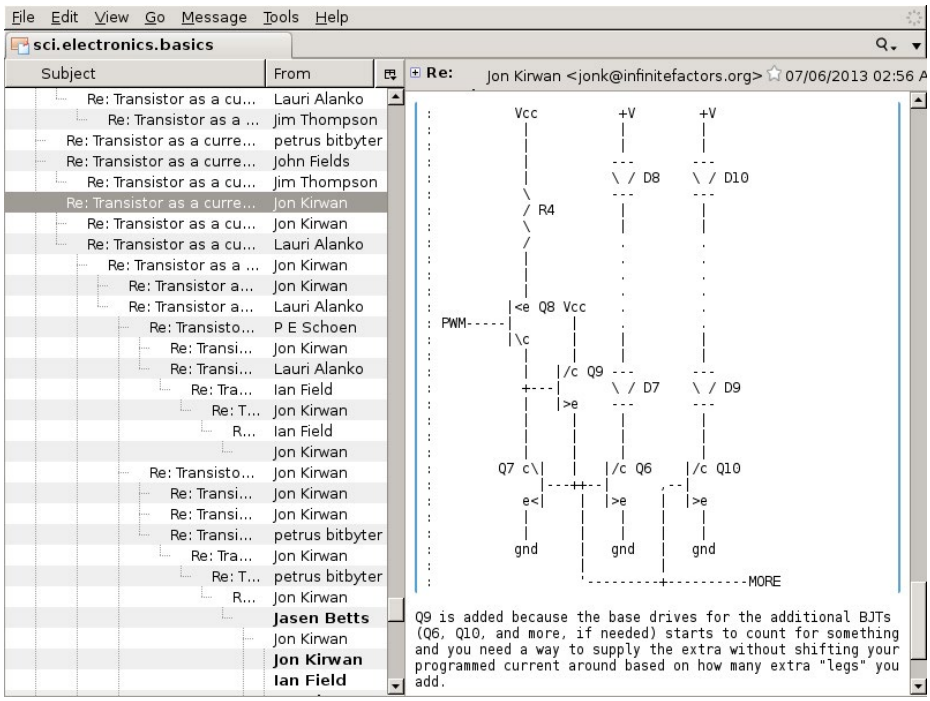
1980-luvulla nopeat tietoliikenneyhteydet yleistyivät. Usenetiä varten luotiin NNTP-protokolla, jonka avulla viestejä voitiin lukea ja lähettää internetin kautta. Pian nyysseistä tuli orastavan internetin vallitseva julkinen keskustelujärjestelmä. Palvelimia syntyi tuhansittain ympäri maailmaa, ja ne levittivät tuhansia keskusteluryhmiä kaikenlaisista aiheista. Kaikki palvelimet olivat tasa-arvoisia, eikä ollut mitään keskustahoa, joka olisi voinut hallita koko verkkoa. Usenetiä voikin pitää varhaisena vertaisverkkona, vaikka yksittäiset käyttäjät eivät yleensä pitäneet omaa palvelinta.

Vaikka nyyssien suosio on vähentynyt, on järjestelmä yhä olemassa, ja se toimii pitkälti samalla tavalla kuin kymmeniä vuosia sitten. Harva verkon teknologia on kestänyt aikaa yhtä hyvin.

Karua mutta toimivaa

Nyyssien seuraamiseen tarvitaan lukuohjelma, joka ottaa yhteyden NNTP-palvelimeen. Lukuohjelmia on niin tekstipohjaisia, graafisia kuin selainkäyttöisiä, ja niitä löytyy kaikille alustoille. Myös monet sähköpostiohjelmat ja jotkin www-selaimet tukevat nyyssejä suoraan. Vaikka erillisen ohjelman käyttöönotto vaatii hieman vaivaa, se maksaa itsensä pian takaisin.

Nyyssien perustoiminnallisuus on vaatimatonta. Viestit ovat yleensä paljasta tekstiä ilman korostuksia tai kuvia. Niitä ei voi muokata jälkikäteen eikä



ASCII-taide taipuu monimutkaisiinkin tarpeisiin. Ohjelma: Mozilla Thunderbird.

siirtää toiseen ryhmään. Mitään erillisiä yhteisöllisiä toimintoja tai profiileja ei ole, vaan järjestelmä keskittyy pelkkään viestimiseen. Lähettäjillä on vain nimi ja osoite, ja joskus harvoin pikkuriikkinen kuvake.

Usenet loistaa siinä, mihin se keskittyy. Monimutkaistenkin viestiketjujen läpi on helppoa suunnistaa hyvällä ohjelmalla. Viestit aukeavat silmänräpäyksessä, sillä pelkkä teksti ei vaadi kaistaa eikä konetehoa. Visuaalisen kikkailun puute auttaa keskittymään itse asiaan, eikä mainoksia järjestelmän ympärille ole edes mahdollista ympätä. Kuvia voi aina linkata webin kautta, ja kaavioiden esittäminen onnistuu monesti perinteisellä ASCII-taiteella.

Nyyssien keskustelujärjestelmä on lähtökohtaisesti avoin. Vaikka palvelimet usein rajoittavat käyttäjäkuntaansa, keskusteluryhmät eivät vaadi erillistä rekisteröitymistä. Kaikki Usenetin käyttäjät voivat lukea ja kirjoittaa kaikissa palvelimen tarjoamissa ryhmissä. Uusiin ryhmiin tutustumiseen on siis erittäin matala kynnys.

Minkä nyyssit menettävät näyttävyydessä, ne voittavat tehokkuudessa ja käytettävyydessä. Kymmenien ryhmien seuraaminen yhden ohjelman sisältä on verrattomasti vaivattomampaa kuin kymmeneen web-foorumiin osallistuminen. Ja kun verkkokatkon aikaan web on täysin käyttökelvoton, nyyssilukijalla voi selailla esiladattuja viestejä paikallisesti ja jopa vastailla niihin.

Keskustelua maailmalla...

Nyyssiryhmillä on pisteillä jaotellut hierarkiset nimet. Esimerkiksi ”comp.lang.c” on C-ohjelmointikieleen keskittyvä ryhmä. Jaottelu auttaa luokittelemaan ryhmät aiheittain ja estämään nimitörmäyksiä. Säännöllinen nimeämiskäytäntö auttaa myös palvelinten ylläpitäjiä määrittämään, kuinka eri ryhmiä kohdellaan palvelimella. Erityisesti nimen alkuosa eli ryhmän ylähierarkia määrää, mitkä palvelimet sitä jakavat. Näin Usenetissä voi olla myös paikallisia ryhmiä, joita ei tarvitse levittää ympäri maailmaa.

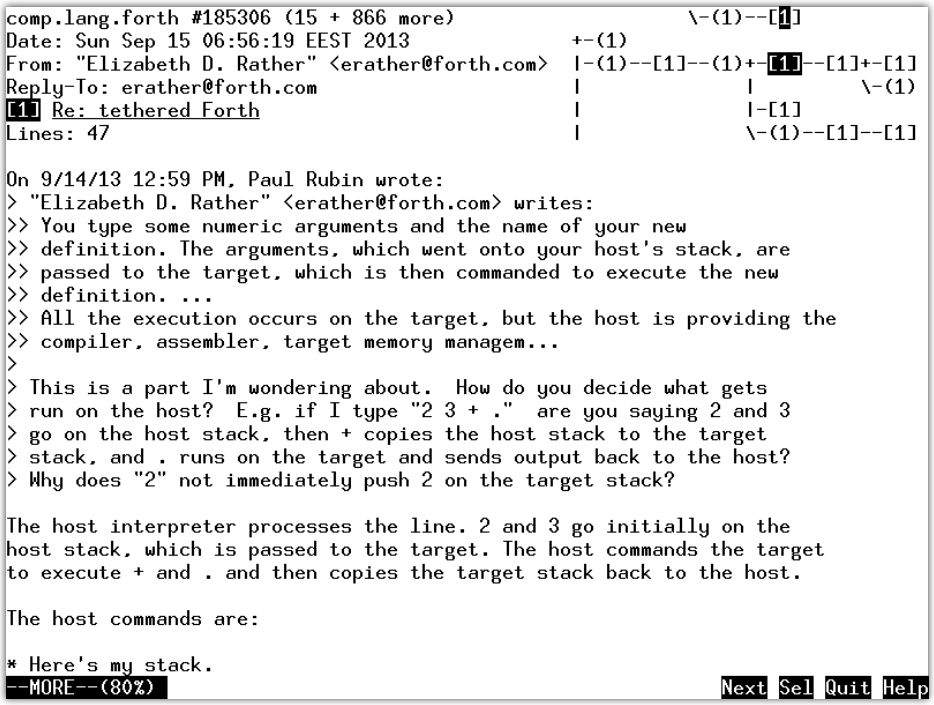
Usenetin ydin koostuu kahdeksasta ylähierarkiasta, joiden ryhmiä kaikkien nyyssipalvelinten odotetaan levittävän. Näitä niin kutsuttuja ”Big-8-hierarkioita” ovat muun muassa ”comp.*” tietotekniikalle, ”rec.*” harrastuksille ja ”sci.*” tieteelle.

Vaikka näihin ryhmiin kirjoittaminen on vapaata, Big-8-hierarkioiden ryhmärakenne on hallinnoitua. Melkein mille tahansa aiheelle voidaan luoda oma ryhmä, mutta sellainen luodaan vain, kun aiheelle on laajaa kiinnostusta. Koska yhdelle aiheelle on aina korkeintaan yksi ryhmä, on Big-8-ryhmien jaottelu varsin siisti. Nämä ryhmät ovat lähtökohtaisesti vilkkaita, vaikka jotkin niistä hiipuvat ajan myötä.

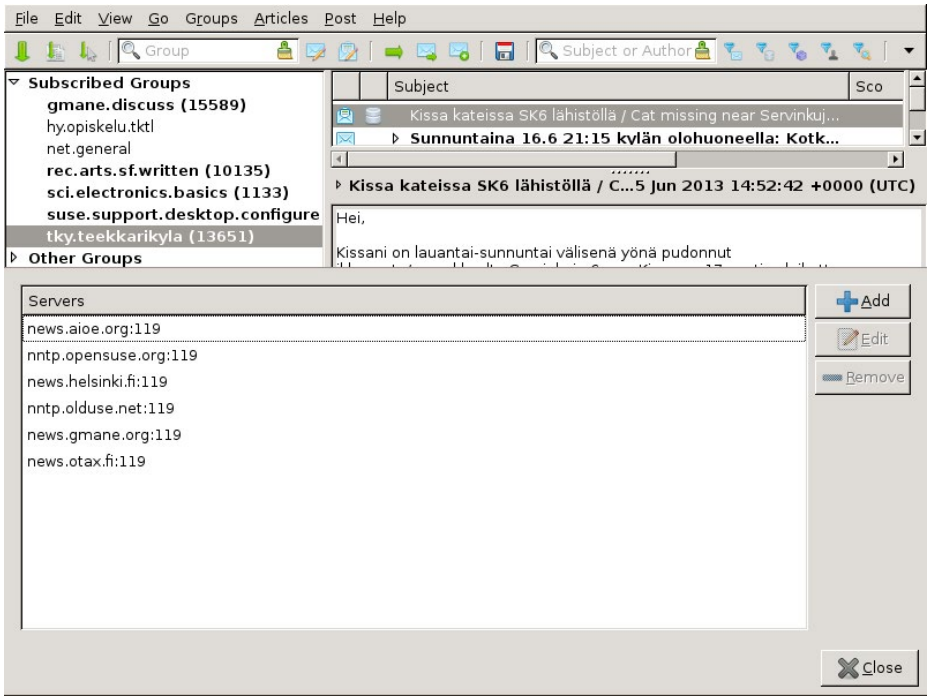
Kaikkia eivät ryhmien hallinnointisäännöt miellytä, ja siksi on olemassa ”alt”-hierarkia, johon kuka hyvänsä voi perustaa uusia ryhmiä ilman byrokratiaa. Koska ryhmien luomiskynnys on niin matala, ne jäävät usein autioiksi tai niiden sisältö ei ole maailmanlaajuisen levityksen arvoista. Tämän vuoksi kaikki palvelimet eivät levitä kaikkia ”alt”-ryhmiä tai säilytä niiden viestejä kovin pitkään.

On myös muunkielisiä keskusteluryhmiä. Monilla kielillä on oma ryhmähierarkiansa, jota jaetaan ainakin kielen levinneisyysalueen palvelimilla, usein muuallakin. Suomenkielisille ryhmille on kaksi hierarkiaa, hallinnoitu ”sfnet” ja vapaampi mutta heikommin levitetty ”finet”. Valitettavasti suomenkielisten ryhmien suosio on vähentynyt roimasti, ja vain muutamassa ryhmässä on enää aktiivista keskustelua, sekin varsin pienen piirin kesken.

Edellisten lisäksi on olemassa monia



Elizabeth Rather, maailman toinen Forth-ohjelmoija, jakaa yhä neuvoja ”comp.lang.forth”-ryhmässä. Ohjelma: trn.



Hyvällä lukuohjelmalla voi seurata niin paikallisia kuin kaukaisiakin keskusteluja. Ohjelma: Pan.

muitakin hierarkioita. Kaikki palvelimet eivät levitä niitä, joten niiden ei katsota kuuluvan varsinaiseen Usenetiin, eikä niillä välttämättä tavoita ihmisiä yhtä laajalti kuin Big-8- ja alt-ryhmillä.

...ja omissa piireissä?

Varsinaisen Usenet-palvelimen perustaminen vaatii sopimusta viestien vaihdosta jonkun toisen palvelimen ylläpitäjän kanssa. Kuka tahansa voi kuitenkin pystyttää oman eristetyn NNTP-palvelimen ja luoda sinne yksityisiä keskusteluryhmiä. Monet korkeakoulut ja yritykset ovat tehneet näin. Tämä on hyvä ratkaisu sisäiseen viestintään pienillekin piireille, sillä monet nyyssiohjelmat pystyvät vaivatta seuraamaan ryhmiä useammasta palvelimesta yhtä aikaa. On siis helpompaa lisätä uusia yksityisiä ryhmiä lukuohjelmaan kuin alkaa seurata uutta web-foorumia säännöllisesti.

Web-foorumit ja nyyssiryhmät eivät ole toisiaan poissulkevia ratkaisuja. Koska nyyssipalvelimen protokolla on avoin, keskusteluille voi tarjota web-käyttöliittymän niille, jotka eivät nyyssejä muuten harrasta. Tällaisia sovelluksia on tarjolla monia, ja onpa tämän jutun kirjoittajakin aikoinaan tehnyt suurehkolle firmalle web-foorumin, jonka alla pyöri nyyssipalvelin. Ylläpito onnistui vakiintuneilla työkaluilla, ja käyttäjät kirjoittivat satoja tuhansia viestejä vuodessa autuaan tietämättöminä alla pyörivästä ikiaikaisesta teknologiasta. Raskaassa käytössä nyyssipalvelimet myös skaalautuvat mainiosti, sillä järjestelmä tukee luonnostaan synkronointia useamman palvelimen kesken.

Nyyssit kuolemaisillaan — jo toistakymmentä vuotta

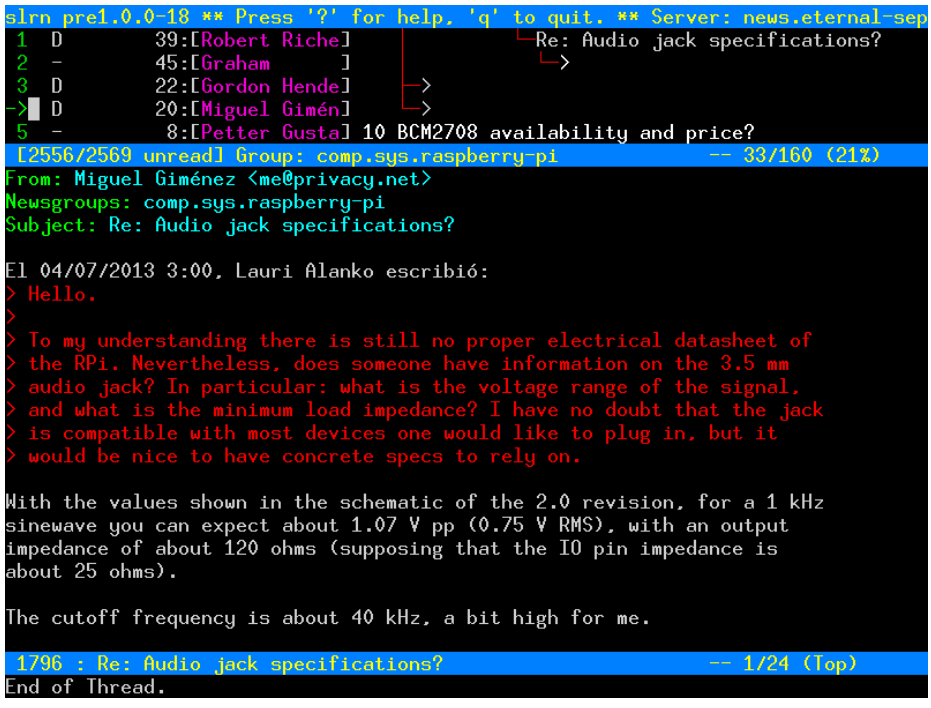
Usenetin tuhoa on povattu jo parinkymmenen vuoden ajan. Joskus syyksi on esitetty keskusteluiden taantumista käyttäjämäärien noustessa, joskus taas pelätty, että viranomaiset ja ylläpitäjät sulkevat koko järjestelmän laittomuuksien estämiseksi. Nykyään pelätään nyyssien muuttumista tiedostonjakovälineeksi ja varsinaisen keskustelun tyrehtymistä. Usenet on silti tähän asti selvinnyt kaikista uhista.

Käyttäjämäärät ovat kuitenkin selvästi laskeneet, ja valtaosa internetin julkisesta keskustelusta on siirtynyt lukemattomiin erilaisiin web-foorumeihin. Yhtenäinen, helppopääsyinen, sulava ja skaalautuva järjestelmä on vaihtunut moninaisten käyttöliittymien, rekisteröintien, selausviiveiden ja ylläpitäjien mielivallan vitsauksiin. Syitä tähän voi vain arvailla.

Yksi syy on eittämättä webin etulyöntiasema. Tavalliset web-haut löytävät kyllä www-sivustojen keskustelut, mutta eivät Usenet-viestejä, ellei joku ole peilannut niitä webiin. Näin ihmiset ohjautuvat web-foorumeille, ennen kuin edes oppivat vaihtoehdon olemassaolosta. Ja vaikka siitä oppisi, erillisen ohjelman asentaminen voi tuntua kohtuuttomalta vaivalta selainkäyttöisten palveluiden rinnalla. Kierre vahvistaa itseään, kun ihmiset hakeutuvat sinne, missä muutkin ovat.

Usenetin ongelma on myös se, että kukaan ei sitä omista. Tämän vuoksi kenelläkään ei ole erityistä intressiä edistää sen käyttöä. Nyyssiryhmän perustamalla ei saa mainetta, mainostuloja kuin sananvaltaakaan, vaan ainoastaan paikan, jossa keskustella. Kaikille tämä ei riitä.

Kaikesta huolimatta nyysseillä on yhä käyttäjänsä. Monissa ryhmissä on vilkasta keskustelua, ja uusia ryhmiä perustetaan yhä. Ehkäpä muutkin vielä saavat jonain päivänä tarpeekseen web-foorumien kömpelyydestä ja rajoitteista, ja löytävät tiensä Usenetiin. Se ei ole lähdössä mihinkään.



Hiljattain perustetussa ”comp.sys.raspberry-pi”-ryhmässä saa vastauksia teknisiinkin kysymyksiin Raspberry Pista. Ohjelma: slrn.

Kuinka sinne pääsee

Nyyssien lukemiseen tarvitsee lukuohjelman ja pääsyn nyyssipalvelimelle. Ilmaisia lukuohjelmia löytyy kaikille alustoille,

ja nyyssipalvelin on perinteisesti ollut sähköpostiin ja kotisivutilaan verrattava peruspalvelu, joka on tullut internet-yhteyden mukana. Nykyään yhteydentarjoajat ovat kuitenkin pitkälti luopuneet nyyssien tarjoamisesta, joten Usenetiin pääsy on hieman hankalampaa. Ilmaisia palvelimia on kuitenkin yhä olemassa.

Lue lisää käytännön ohjeita Usenetiin pääsemiseen jutun verkkoliitteestä Skrollin sivuilla!
http://skrolli.fi/2013.4/usenet

Binääriryhmät: nyyssien turmio?

Usenetin alkuaikoina verkkoyhteydet olivat vaatimattomia ja tiedostojen saatavuus verkon kautta oli heikkoa. Tavallisilla käyttäjillä ei ollut tiedostonjakopalvelimia. Koska Usenet kuitenkin tavoitti ihmisiä laajalti, ryhdyttiin sitä käyttämään tiedostojenkin levitykseen. Se oli monille ainoa tapa tarjota omia tiedostoja yleiseen käyttöön. Tiedostojen levitykseen luotiin erilliset ”binääriryhmät”, joita monet palvelimet kohtelivat eri tavalla kuin tavallisia keskusteluryhmiä.

Käyttäjäkunnan kasvaessa myös binääriryhmien suosio kasvoi, ja ne alkoivat muodostaa valtaosan nyyssipalvelimien kuormituksesta. Usenet ei ollut erityisen tehokas tapa levittää tiedostoja, joten www:n ja vertaisverkkopalvelujen kehittyessä binääriryhmien olisi pitänyt vaipua unholaan vanhanaikaisina.

Toisin kuitenkin kävi. Monet tiedostoja laittomasti jakavat alkoivat pelätä viranomaisten jäljitystä, joten nyyssien binääriryhmien epäsuoruus vaikutti houkuttelevalta: koska kaikki liikenne kulkee nyyssipalvelimen kautta, toimii luotettu palvelin anonymisoijana, joka piilottaa, kuka on tiedoston lähettänyt, ja ketkä sen ovat ladanneet. Onkin syntynyt monia NNTP-pohjaisia tiedostonjakopalvelimia, jotka tarjoavat maksullista palvelua innokkaimmille lataajille ja jakajille.

Massiivisen tiedostonjaon vuoksi ilmaiset, keskustelupohjaiset palvelimet eivät enää tue binäärejä ollenkaan, vaikka joskus sellaisille olisi keskustelujen lomassa perusteltua tarvetta. Nyyssien maine laittomuuksien välineenä on myös saanut jotkut tahot sulkemaan palvelimensa kokonaan. Ja koska tiedostonjakajat mainostavat välinettään nimellä ”Usenet”, on sana saanut verkossa uuden merkityksen, jonka alta ei alkuperäistä keskustelujärjestelmää enää helposti löydä.

Vaikka binääriryhmien suosio onkin osoitus nyyssiteknologian monikäyttöisyydestä, on se tehnyt vain hallaa todelliselle Usenetille.

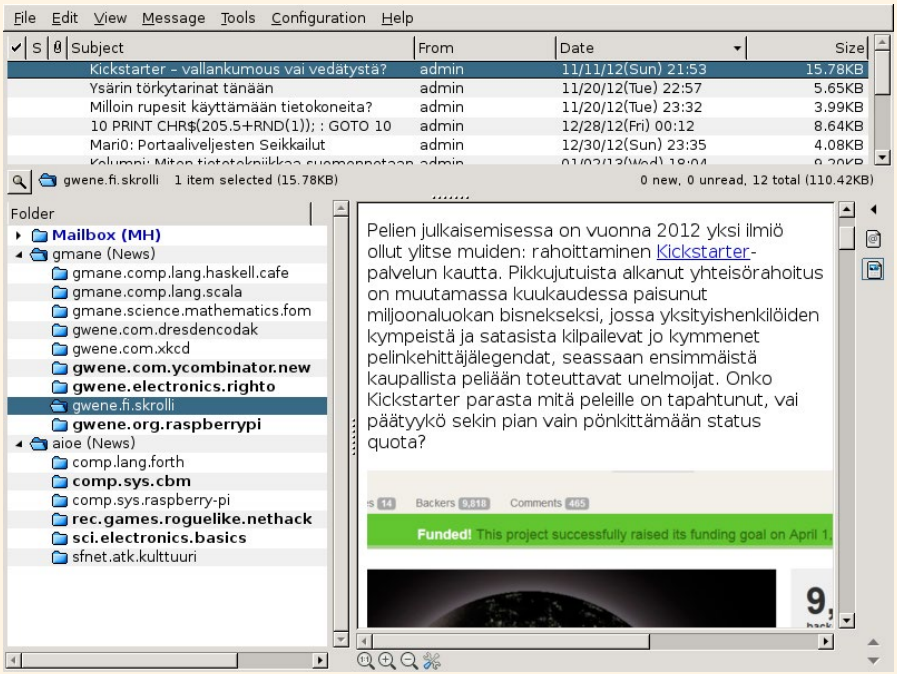
Gmane ja Gwene – maili ja web nyysseihin

Sähköpostilistat ovat nyyssien ohella toinen perinteinen verkon massaviestintäväline. Postituslistoja on kuitenkin vaivalloista tilata ja lopettaa, ja saapuvien listaviestien lajittelu vaatii oman vaivansa. Webissä on kyllä monien listojen arkistoja, mutta keskustelujen seuraaminen niiden kautta on hankalaa.

Gmane on palvelu, joka ratkaisee nämä ongelmat tarjoamalla monia julkisia postituslistoja nyyssiryhmien kautta. Gmanen NNTP-palvelimen kautta pääsee lukemaan ”gmane”-hierarkian ryhmiä, jotka peilaavat eri postituslistoja. Näin nyyssien käyttäjän ei tarvitse kuin lisätä Gmanen palvelin ohjelmaansa, jotta saa pääsyn tuhansiin uusiin keskustelualueisiin varsinaisen Usenetin lisäksi.

Gmanella on myös sisarpalvelu Gwene, joka ohjaa webin RSS-syötteitä nyyssiryhmiin. Sen avulla monia blogeja ja foorumeitakin voi lukea nyyssiohjelman kautta, joten säännöllinen verkkopäivitysten seuraaminen nopeutuu.

- http://gmane.org/
- http://gwene.org/



Skrolli.fi nyysseissä! Ohjelma: Claws Mail.

Internetit ennen Internetiä

– modeemipurkkien nousu ja tuho

Suomi on ollut nyt 25 vuotta kiinni Internetissä. Tietokoneet ovat kuitenkin viestineet toisilleen jo paljon tätä ennen, ja Internetilläkin meni vuosikausia, ennen kuin se syrjäytti muut verkot.

Teksti: Ville-Matias Heikkilä Kuvat: Manu Pärssinen, Ville-Matias Heikkilä, Wikimedia Commons

Koneetkin puhuivat puhelimessa

Sähköisiä tietoverkkoja ruvettiin rakentamaan jo 1800-luvulla, ja eräs niistä levittäytyi myöhemmin lähes joka kotiin. Kyseessä oli tietenkin puhelinverkko. Suomalaiset olivat alusta asti innostuneet verkon rakentamisesta: 1890-luvulla Helsingissä oli jo useita puhelimia sataa asukasta kohden, mikä oli kärkilukuja koko maailmassa.

Puhelinverkko oli houkutteleva ratkaisu myös muun kuin puheen välittämiseen. Uutistoimistojen piti lähettää paljon tekstiä toisilleen nopeasti, ja siihen käytettiin kaukokirjoittimia, eräänlaisia etäohjattavia sähkökirjoituskoneita. Kaukokirjoittimilla oli oma Telex-verkko, mutta puhelinverkko ulottui laajemmalle ja oli usein edullisempi. Kirjoittimen ja puhelinlangan väliin tarvittiin vain modeemi eli modulaattori-demodulaattori, joka muuntaa merkkidataa äänitaajuuksiksi ja takaisin. Ensimmäiset puhelinmodeemit otettiin käyttöön 1940-luvulla, joten alusta asti oli valmiina tekniikka tietokoneiden kytkemiseksi puhelinverkkoon.

Ensimmäinen laaja tietokoneverkko oli vuonna 1958 jenkkiarmeijan käyttöön ottama SAGE (Semi-Automatic Ground Environment). Se käytti yhteydenpitoon enimmäkseen tavallisia puhelinyhteyksiä ja modeemeja. Siviilipuolella varhaisia omaksujia olivat pankit, joiden keskustietokoneet soittelivat toisilleen vaihtaakseen rahaliikennetietoja.

Hyvästit reikäkorttipinoille

Tietokoneet olivat alkuvuosikymmeninään isoja ja kalliita, ja harva pääsi käyttämään niitä henkilökohtaisesti. Käyttäjät suunnittelivat ohjelmat paperilla, lävistivät ne reikäkorteille, toimittivat korttipinot konesaliin eräajettavaksi ja saivat ohjelman tulosteet aikanaan itselleen. Myöhemmin käyttötavat muuttuivat dramaattisesti, kun koneisiin alettiin kytkeä päätteitä. Ositusajossa useammat käyttäjät komensivat samaa keskustietokonetta samanaikaisesti eri päätteiltä ja saivat vasteet liki välittömästi. Päätteet liitettiin usein modeemeilla, joten käyttäjä saattoi olla eri kaupungissa kuin tietokone.

Tietokoneet, päätteet ja modeemit kiinnostivat myös tekniikkaharrastajia, ja monet rakensivat niitä jopa itse. Tekniikkaan paneutunut opiskelija saattoi näin käyttää koulunsa keskuskonetta kotoa käsin. Ennen pitkää sikisi myös ajatus tehdä omasta kotitietokoneesta ”keskuskone”, johon toiset voisivat ottaa yhteyksiä.

Maailman ensimmäisen julkisen harrastaja-BBS:n pystyttivät chicagolaiset Ward Christensen ja Randy Suess helmi-

kuussa 1978. Järjestelmä oli nimeltään yksinkertaisesti CBBS (Computerized Bulletin Board System) ja se toimi Altair-yhteensopivalla koneella. Suomen ensimmäisen vastaavan järjestelmän perusti kesäkuussa 1982 Seppo Uusitupa, ja se oli nimeltään CBBS Helsinki.

Purkkien toiminta

BBS-järjestelmiä kutsuttiin Suomessa 1980-luvulla sähköisiksi postilaatikoiksi tai bokseiksi mutta myöhemmin myös kannuiksi ja purkeiksi. Purkkien toimintaa ja viehätystä oli niiden kulta-aikana vaikeaa selostaa maallikoille, mutta nykyisin se onnistuu vertaamalla niitä WWW-sivustoihin.

Kuvittele sivusto, jolla on keskustelufoorumi. Foorumilla kirjoittelee useita kymmeniä ihmisiä kaikesta mahdollisesta. Foorumin lisäksi sivustolla on alue, josta saa ladattua ohjelmia ja muita tiedostoja ja jonne voi lähettää omia tiedostojaan. Lisäksi omassa osiossaan voi esimerkiksi pelata pelejä muita käyttäjiä vastaan. Sivustolle kirjaudutaan käyttäjätunnuksella ja salasanalla, joiden saamiseksi on vastailtava muutamiin kysymyksiin, lähinnä annettava henkilötietoja.

Tällainen sivusto on helppo käsittää useimmille Internetin käyttäjille. Kuvittele seuraavaksi, että sivustoa pystyy käyttämään vain yksi käyttäjä kerrallaan. Kun sivustolle yrittää päästä, se on todennäköisesti varattu. Se vapautuu ehkä vartin tai tunnin kuluttua. Odotellessa voi yrittää päästä jollekin toiselle sivustolle. Kun jonnekin vihdoin pääsee sisään, on yhteysaikaa käytettävissä rajoitetusti, esimerkiksi puoli tuntia. Lisäksi yhteys maksaa vähintään paikallispuhelumaksun.

Yhteysaika kannattaa käyttää tehokkaasti. Foorumiviestejä tai tiedostolistoja ei kannata jäädä selailemaan, vaan ne on

Kaukokirjoitin.

hyvä ladata omalle koneelle tutkittavaksi. Viestien vastaukset kannattaa kirjoittaa omalla koneella erityisellä offline-lukuohjelmalla ja lähettää ne pakettina takaisin sivustolle vaikka seuraavana iltana.

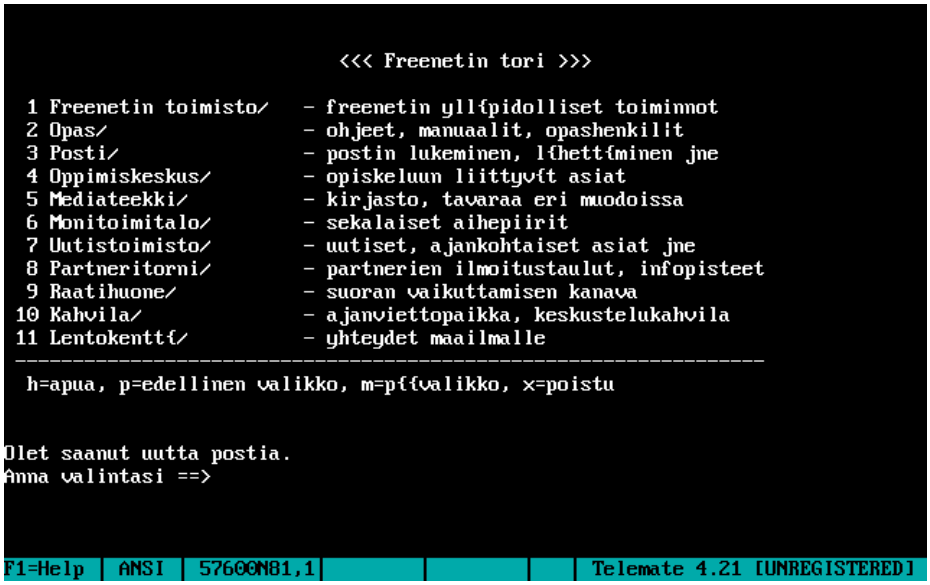
Tietenkään BBS-purkit eivät olleet mitään sivustoja, vaan yhteystapana oli suora, merkkipohjainen pääteyhteys. Mustalle pohjalle ilmestyi verkkaisesti listoja komennoista, joita kirjoittamalla pääsi liikkumaan järjestelmässä alueelta toiselle ja tekemään eri asioita. Purkin palvelinkone sijaitsi useimmiten ylläpitäjänsä, systeemioperaattorin eli SysOpin kotona. Ylläpitäjä seuraili usein käyttäjiensä touhuja ja saattoi tulla välillä chattailemaan kahden kesken. Purkit oli usein sisustettu persoonallisiksi värikkään merkkigrafiikan avulla, ja niihin soittaminen tuntui parhaimmillaan vierailulta omistajansa kotiin.

Isompiakin purkkeja toki oli. Monet olivat aivan oikeissa laitetiloissa, monilinjaisia ja joskus jopa maksullisia. Suomen suurin purkki oli vuosina 1994–2002 toiminut MBnet, jossa oli tilaa yli 500 yhtäaikaiselle käyttäjälle. Tyypillinen purkki toimi kuitenkin harrastajan kotikoneessa. Osa oli auki ympäri vuorokauden, osa vain öisin.

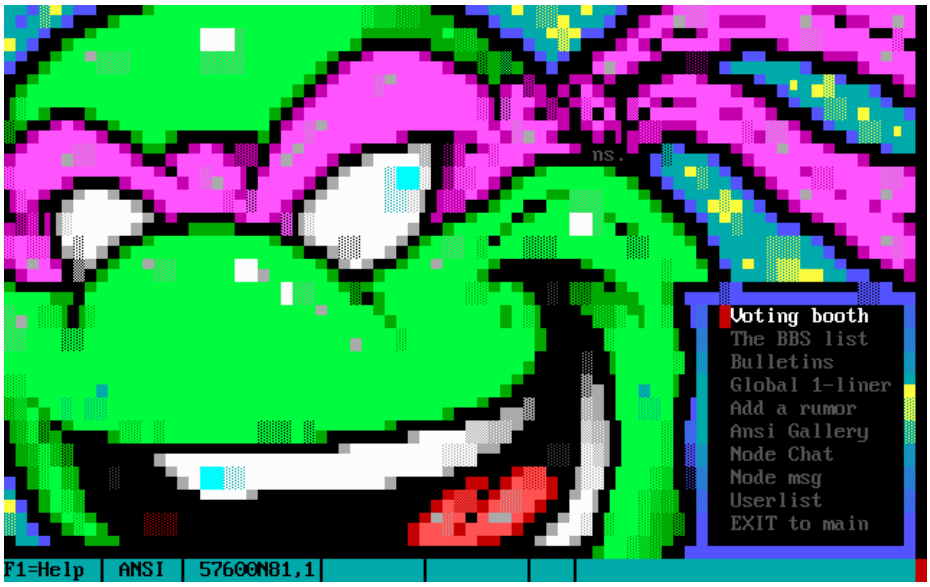
Purkkeihin soittelu ei vielä 1980-luvulla ollut kovin yleistä suomalaisten tietokoneharrastajien keskuudessa. Jopa piraatit vaihtelivat ohjelmakopioitaan aina vuosikymmenen lopulle asti etupäässä postitse, levykkeillä. Osasyynä oli varmasti se, että modeemin käyttöönotto oli tsaarinaikaisen lennätinlain vuoksi pitkään harrastajille melko kallista.

BBS-tyylistä viestintää haluttiin markkinoida myös massoille: teksti-tv:n serkkuna 1970-luvun lopulla syntyneen videotexin piti mahdollistaa joka kodin tietoverkkoyhteydet helppokäyttöisten päätelaitteiden avulla. Suomalaisia videotex-palveluja olivat Telesampo ja Infotel, joita käytettiin etenkin pankkiyhteyksiin. Järjestelmä löi kuitenkin kunnolla läpi vain Ranskassa, jonka Minitel-

BBBS-ohjelmistoa käyttävän purkin valikkoja.



Freenet Finland oli monille purkkikäyttäjille ensikosketus Internetiin.



Haciend el Bananasin ansi-grafiikkaa.

Eräs suosituimpia purkkipelejä, Legend of the Red Dragon eli LORD.

palvelulla oli parhaimmillaan miljoonia käyttäjiä.

Isot koneet ja yliopistot

Samaan aikaan, kun harrastajat rakensivat ensimmäisiä modeemiboksejaan, keskuskoneiden maailmassa rakennettiin jo pakettiverkkoja kiinteiden yhteyksien varaan. Yhdysvaltalaiset olivat vuodesta 1969 lähtien rakentaneet Arpanetiä, johon kuului puolustushallinnon ja yliopistojen koneita. Euroopassa vastaava verkko oli ranskalaisten Cyclades, ja hahmoteltiinpa Suomeenkin samantapaista yliopistoverkkoa jo vuonna 1974.

Kiinteät linjat olivat fyysisesti samanlaisia kuparijohtoja kuin puhelinlangatkin, mutta ne oli kokonaisuudessaan varattu kahden pisteen yhdistämiseen. Asiakas maksoi linjasta kiinteää vuokraa ja sai sen koko kapasiteetin omaan käyttöönsä. Arpanetin linjoilla siirrettiin dataa aluksi 50 kilobitin sekuntinopeudella – parikymmentä kertaa nopeammin, kuin mihin tavallisella puhelinmodeemilla pääsi.

Pakettiverkon ideana on, että verkkoon kytketyt koneet lähettävät linjalle datapaketteja, joihin kuuluu vastaanottavan koneen osoite. Koneet myös kuuntelevat linjaa jatkuvasti ja ottavat talteen itselleen osoitetut paketit. Linjojen risteyksissä on reitittimiä, jotka ohjaavat paketteja osoitteen mukaan eteenpäin. Arpanet käytti aluksi NCP-nimistä paketointikäytäntöä mutta vaihtoi vuonna 1983 Internet-protokollaan eli IP:hen. Samalla armeijan koneet erkanivat omaksi verkokseen, ja jäljelle jäänyttä osaa alettiin kutsua nimellä Internet.

Suomessa Internet-tyyppinen viestintä alkoi kuitenkin täysin modeemivetoisilla verkoilla. Yliopistoilla oli 1980-luvun alussa Unix-koneita, jotka tukivat käyttäjien välistä sähköpostia (mail) ja julkisia keskustelualueita (news). Unixissa oli ohjelma nimeltä Unix to Unix Copy (UUCP), jonka avulla Unix-koneet pystyivät siirtämään keskusteluja toisilleen modeemiyhteyksien yli. Postisäkkejä vaihdettiin yleensä vain kerran pari päivässä, joten laajemmissa UUCP-verkoissa viestit matkasivat päivätolkulla. Suomen verkko yhdistettiin kansainväliseen UUCP-verkkoon vuonna 1983. BBS-maailmassa myöhemmin yleistynyt Fidonet perustui myös viestipakettien vaihtamiseen.

Pakettiverkot tulivat Suomeen vuonna 1983, kun Posti- ja telelaitos alkoi tarjota Datapak-nimistä X.25-verkkoa. Laskutus perustui siirretyn datan määrään. Verkkoa käyttivät aluksi etenkin pankit, mutta myös samana vuonna alulle pantu Suomen yliopistojen Funet-verkko päätti ottaa sen käyttöönsä. Funet siirtyi kuitenkin joidenkin vuosien päästä kiinteisiin 64 kilobitin vuokralinjoihin, kun runsas sähköpostiliikenne aiheutti sille satojentuhansien markkojen laskuja.

Yliopistoissa oli monien eri valmistajien koneita, joilla oli kaikilla eri verkkotekniikat, ja Funetissa liikkui kaikkien paketteja sekaisin. Decnetissä oli vain VAXeja ja muita Digitalin koneita. EARN oli IBM:n perustama ja sponsoroima eurooppalaisten yliopistojen verkko, jonka hienouksia olivat postituslistat ja RELAY-pikaviestit. Unix-koneet puhuivat vain IP:tä, mutta VMS-koneet puhuivat kaikkia protokollia. Ainoa kiinteä ulkomaanyhteys oli EARNin kautta, joten kansainväliset Unix-sähköpostit ja -nyyssit jouduttiin vielä vuonna 1987 siirtämään modeemeilla. Tästä päästiin eroon seuraavana vuonna, kun Pohjoismaiden yliopistoverkot yhdistettiin 64 kilobitin linjoilla Nordunet-verkoksi.

Nordunet yhdistyi Internetiin marraskuussa 1988, kun Tukholman ja Princetonin välille avattiin 56 kilobitin satelliittiyhteys. Yhdysvallat oli kuitenkin varuillaan Suomen suhteen, olihan se aivan Neuvostoliiton kupeessa. Vielä saman vuoden puolella Pohjoismaiden Internet-yhteys pistettiin poikki viikoksi, kun joku innokas suomalaisopiskelija oli kolkutellut jenkkiarmeijan koneita.

Internet syrjäytti vähitellen muut protokollat yliopistoverkossa. Nykyään harva tunnistaa esimerkiksi EARNia nimeltä, vaikka se oli monen vuoden ajan suosituin tietoverkko Euroopan yliopistoissa.

Maailmojen kilpajuoksu

Purkit olivat vielä 1990-luvun alussa selvästi eri maailmassa kuin pakettiverkot. Useimpien BBS-käyttäjien käsitys Internetistä olikin melko epämääräinen. Joillakin purkeilla oli jonkinlainen yhteys Internetin puolelle esimerkiksi UUCP-

PTT BBS, jättimäinen taiwanilaispurkki.

verkon kautta, ja olipa Suomessa jo pari kiinteästi nettiin kytkettyä BBS:ää yliopistojen tiloissa.

Monet harrastajat ottivat ensimmäiset Internet-yhteytensä noin vuonna 1993. Tällöin Suomeen ilmestyi lukuisia nuorison saavutettavissa olevia palveluntarjoajia. Useat näistä olivat itsekin purkkeja, joiden kiinteä Internet-linja kustannettiin käyttäjien maksamilla muutaman kymmenen markan kuukausimaksuilla. Käyttöliittymät olivat edelleen merkkipohjaisia, käyttöajat rajoitettuja ja linjat tuuttasivat usein varattua.

Useimmille Internetin palveluille oli tuttu vastine BBS-maailmassa. Sähköposti vastasi yksityisviestejä, nyyssit julkisia viestejä ja IRC monilinjaisten purkkien chatteja. Mittakaava oli kuitenkin aivan toinen. Vilkkaita keskustelualueita oli tuhansia, ja niitä saattoi löytää hyvin kapeille ja erikoisille aihepiireille. Chateissa ja MUD-peleissä oli käsittämätön määrä samanaikaisia käyttäjiä kymmenistä eri maista, eivätkä valtioiden rajat tuntuneet missään. Olo oli kuin maalaiskylän kasvatilla suurkaupungin keskellä.

Modeemit yleistyivät 1990-luvulla, mikä näkyi sekä BBS- että Internet-käytön kasvuna. Monet käyttivät ahkerasti molempia, mutta eri tarkoituksiin. Purkit koettiin selkeinä ja tiiviinä pienyhteisöinä, joissa samanhenkiset ihmiset jutustelivat kaikesta mahdollisesta keskenään. Paikallispuhelualueiden rajat synnyttivät kuhunkin maakuntaan oman purkkiskenen, jonka jäsenet tapasivat tosiaan myös elävässä elämässä. Internet oli kasvottomampi, kaoottisempi ja joka suuntaan rajaton ”tiedon valtameri”, josta haettiin erityisaiheisiin liittyviä tiedostoja, keskusteluja ja asiantuntemusta.

Vuosikymmenen jälkipuoliskolla netti alkoi syödä BBS-maailmaa. Alkuvuodesta 1996 oli Suomessa julkisia 24H-purkkeja vielä yli 500, mutta vuonna 2000 enää joitakin kymmeniä. Kiinteät laajakaistayhteydet saivat monet ahkerat käyttäjät luopumaan lankapuhelinliittymistään ja siten myös purkeista. Uskollisimmat BBS-intoilijat lähtivät vasta pakon edessä, eikä siirtymä ollut helppo.

Monet purkkiyhteisöt hajosivat kokonaan, koska netin puolelta ei löytynyt sopivaa kokoontumispaikkaa. Sitkeimmät systeemioperaattorit siirsivät purkkinsa Internetin puolelle joko sellaisinaan tai nettifoorumeiksi muutettuina, mutta harva menestyi enää tämän jälkeen. Parhaiten koossa pysyivät ne yhteisöt, jotka vaihtoivat purkit IRC-kanaviin. IRC oli toki olemukseltaan hyvin erilainen, mutta se onnistui vangitsemaan purkeille ominaisen tiiviin yhteisöllisyyden paremmin kuin esimerkiksi foorumit tai nyyssiryhmät.

Ikuisen syyskuun loppua odotellessa

Internet on Suomessa olonsa aikana kasvanut osaajaeliitin salaseurasta koko kansan perusoikeudeksi. Tutkijoiden, opiskelijoiden ja tietokoneihmisten perässä seurasivat vähitellen kaikki väestökerrokset. Koukuttava nettiyhteisöllisyys tuli Pihtiputaan mummollekin tutuksi viimeistään Facebookin myötä.

Internet on muuttunut vuosien mittaan, välillä jopa parempaan suuntaan. Äänensä kuuluviin saaminen on nykyisin helppoa myös teknisesti tumpelommille, ja kiinnostavan sisällön löytäminen kohinan keskeltä on vaivattomampaa. Muutos pakottaa kuitenkin myös luopumaan: kun suositun nettipalvelun kukoistus loppuu ja käyttäjät siirtyvät toisaalle, ei samanlainen yhteisöllisyys enää toistu. Uudesta palvelusta puuttuu aina jokin ratkaiseva ominaispiirre, eivätkä monet sopeudu uuteen. Yhteisöt hajoavat, ja monet jäävät kaipaamaan entistä.

Ensi vuosikymmenellä Facebook on jo vaihtunut johonkin aivan muuhun. Viimeistään siinä vaiheessa suuretkin kansanjoukot ovat alkaneet ymmärtää Internetin historiallisuutta – sitä, kuinka tietoverkkokulttuurissakin on murroksia ja uusi rakentuu aina vanhan päälle. Ehkäpä silloin myös BBS-historia voidaan nostaa ansaitsemaansa arvoon, kukapa tietää?

Purkkitilanne 2013

Sivusto telnetbbsguide.com listaa tätä kirjoitettaessa 350 toiminnassa olevaa BBS:ää. Valtaosaan saa yhteyden vain telnetillä, mutta muutamalla amerikkalaisboksilla on myös puhelinlinja. Listalla on myös kolme suomalaista purkkia: BBBS-ohjelmiston tukipurkki BCG-Box (bbbs.net), Rampton Bird’s Box (rbb.bbs.fi:32) ja tietokonetaiteeseen erikoistunut Haciend El Bananas (haciend.bbs.fi).

Monet nykyboksit ovat varsin autioita – kuin museoesineitä, jotka on jätetty esille, vaikka käyttäjät ovat kadonneet. On kuitenkin edelleen myös aktiivisia ja hyvin suuria järjestelmiä. Taiwanilaisessa PTT:ssä on parhaimmillaan yli 150 000 yhtäaikaista käyttäjää.

Länsimainen BBS-skene on läheisessä suhteessa MUD-peleihin, eikä purkin ja mudin raja ole aina edes kovin selvä. Mudit voivat kuitenkin edelleen hyvin wowien ja lolien noususta huolimatta: suurimmissa mudissa on mudstats.comin mukaan parhaimmillaan yli 800 yhtäaikaista pelaajaa. Suomalaista Batmudia pelaa enimmillään noin 200.

Lähteitä ja muuta mediaa:

- Funet – Suomen tie Internetiin (CSC:n julkaisu, 2008)
- BBS-harrastajat 1990-luvun tietoverkkokulttuurin murrosvaiheessa (Mikko Hirvonen 2010, pro gradu, Digitaalisen kulttuurin laitos, Turun yliopisto)
- Funetista Facebookiin: Internetin kulttuurihistoriaa (Helsingin yliopisto, 2009)
- BBS: The Documentary, amerikkalaisen Jason Scottin dokumenttisarja

Kartan lähde: International Telegraph Bureau, Bern, Sveitsi, 1901. Jäljennöksen luovuttanut käyttöön Bostonin yleisen kirjaston Norman B. Leventhal Map Center.

Lennätin – 1800-luvun netti

Sana *lennätin* tuo useimmille mieleen lähinnä morseaakkoset ja sähköisen piipityksen. Kulttuurisessa mielessä varhainen viestijärjestelmä sisälsi kuitenkin tuttuja nykyajan ilmiöitä. Silloinkin läheteltiin statuspäivityksiä ja tuskailtiin salakuuntelun vuoksi.

Teksti: Ronja Koistinen Kuvat: Ignacio Cobos Rey, Wikimedia Commons, Flickr-käyttäjä Groume

TeliaSonera ja Itella lukeutuvat Suomen viestinnän infrastruktuurille tärkeimpien yritysten joukkoon. Välillä on hyvä palauttaa mieleen, että vaikka yhtiöiden toimialat eivät paljoa kohtaa, niillä on yhteinen menneisyys. Sonera oli ennen Tele ja Itella oli Posti, mutta sitä ennen ne olivat valtion liikelaitos nimeltä Posti- ja telelaitos. Vielä aiemmin, vuoteen 1981 asti, laitoksen nimi oli Posti- ja lennätinlaitos.

Tämä artikkeli ei ole historiikki valtion omistusten yksityistämiskehityksestä, mutta organisaatioiden kautta kehityksen käänteisiin on helppo sukeltaa nykyisyydestä käsin. Aikaamme leimaa internet ja sen syövereissä vellova kulttuurin ja viestinnän meri. Meemit ja omaleimaiset sukupolvikokemukset puhuttavat.

Sähköisten viestien välittämistä globaalin mittakaavan verkossa ei kuitenkaan keksitty 1980-luvulla. Jo edelliselle vuosisadalla kukoisti maapallon joka kolkkaan levittäytynyt, yötä päivää piipittävä ja nakuttava sähköinen viestijärjestelmä, lennätinverkko.

Mikä lennätin?

1860-luvulle tultaessa oli olemassa maailmanlaajuinen lennätinjärjestelmä, verkosto piuhoja ja viestikeskuksia, joita pitkin operaattorit lähettivät sähkösanomia paikasta toiseen. Viestin lähettääkseen asiakas käveli konttoriin ja kirjoitti paperikaavakkeelle haluamansa viestin ja vastaanottajan katuosoitteen. Sähkeen hinta määräytyi viestin kulkeman matkan perusteella.

Sähköttäjä lähetti viestin eteenpäin lennätinlaitteella muuntamalla sen morseaakkosten mukaiseksi ”pisteiden” ja ”viivojen” muodostamaksi koodiksi. Sähkeet reititettiin käsin karttojen ja taulukoiden avulla keskustoimistojen kautta, ja käytyään läpi tarvittavat hypyt viesti pääsi lopulta perille vastaanottajalle.

Sähköistä lennätintä edelsi Napoleonin Ranskassa 1700-luvulla keksitty optinen lennätin. Ranskalainen Claude Chappe kehitti järjestelmän, joka koostui korkeille mäille rakennetuista torneista. Niissä oli jatkuva miehitys. Tornit muistuttivat vähän tuulimyllyjä, sillä niillä viestittiin liikuttamalla merkinantovipuja pitkien sauvojen päässä. Merkit luettiin seuraavalta tornilta kaukoputken avulla ja viesti välitettiin eteenpäin seuraavalle tornille.

Optisia lennätinketjuja oli pian Keski- ja Etelä-Euroopassa useissa maissa. Joissakin ketjuissa lähetettiin viestit suoraan aakkosia merkitsemällä, kun toisissa taas viestittiin numeroita, joiden merkitys luettiin erityisistä koodikirjoista. Kirjojen julkaiseminen olikin pitkään isoa liiketoimintaa.

Myös Suomen ensimmäinen kosketus lennättimiin tapahtui jo 1700-luvulla. Suomi kuului tuolloin Ruotsiin, jossa hääri toimelias vapaaherra Abraham Niclas Edelcrantz. Hän toi lennätinteknologian Ranskasta Ruotsiin. 1790-luvulla Ruotsin itärannikolle ja Länsi-Suomeen rakennettiin optisia lennätintorneja, joista oli Ruotsille suuri hyöty Suomen sodassa 1808–1809 Venäjää vastaan. Ruotsalaisten rakentamia torneja oli ainakin Ahvenanmaalla, mutta venäläiset tuhosivat ne sodan jälkeen. Myös Venäjän ensimmäinen optinen lennätinlinja rakennettiin Pietarin ja Laatokan välille 1824.

Alkuperäisistä optisista lennätintorneista on säilynyt varsin vähän arkeologisia jäänteitä. Ruotsista ja Suomesta ei tiettävästi tunneta raunioita. Sen sijaan monien maiden paikannimistössä on jäänteitä näiden tornien sijainneista. Isossa-Britanniassa on paljon Telegraph Hill -nimisiä, Ruotsissa ja Suomen ete-

Espanjassa säilynyt optisen lennätintornin raunio.

Morsen lyhyisiin ja pitkiin signaaleihin perustuva järjestelmä mullisti sähkötysnopeuden. Yksinkertaisella sähkötysavaimella saattoi taitava sähköttäjä viestiä hyvinkin nopeasti.

lärannikolla taas Telegrafberget-nimisiä kukkuloita. Muutamia suomenkielisiäkin Telegrafivuoria ja -kallioita on siellä täällä.

Operaattori kuulee kaiken

Lennättimessä ja nykyajan internetissä olennainen ero on reitityksessä. Internet reitittyy automaattisesti koneiden avulla, mutta sähkeitä reitittivät ihmissähköttäjät käsityönä. Viestit eivät voineet koskaan olla täysin luottamuksellisia, sillä jokaisessa linkkitoimistossa oli vastassa ihminen. Sähköttäjä kuunteli viestin linjalta, kirjoitti sen paperille ja sähkötti eteenpäin seuraavaan toimistoon kohti viestin määränpäätä.

Myös sotilastiedustelu kiinnostui lennätinliikenteestä melko pian. Ensimmäinen sotilaallinen koodinmurtamiseen erikoistunut organisaatio oli ranskalainen Bureau du Chiffre. Brittiläinen imperiumi olikin huolissaan vieraiden valtojen kuuntelusta ja rakensi oman verkkonsa. Brittien verkosta oli yhteydet muuhun kansainväliseen verkkoon vain tietyistä kohdista, joiden viestiliikennettä valvottiin tarkasti.

Sähköttäjien, kilpailijoiden ja vieraiden valtojen silmien välttämiseksi alettiin käyttää salakirjoitusta. Koodikirjat tulivat uudelleen muotiin, ja kirjapainot alkoivat suoltaa kaupallisia salauskirjoja. Jotkin kirjat olivat koko painokseltaan samanlaisia, mutta monissa oli myös mahdollisuus omiin merkintöihin. Kirjoista saattoi esimerkiksi ennalta sovitusti vaihtaa sivuja keskenään tekemällä sivunumerointiin uudelleenohjauksia. Näin viestin matkalla napannut taho ei voinut purkaa ainakaan koko sisältöä, vaikka olisikin tiennyt, minkä merkkiseen koodikirjaan salakirjoitus perustui.

Valtiot valvoivat ennen pitkää myös kotimaanliikennettä. Syntyi erilaisia yhteistyösopimuksia, ja 1865 perustettiin International Telegraph Union eli ITU. (ITU on olemassa edelleen, joskin sen nimi on vaihtunut muotoon International Telecommunication Union.)

ITU närkästyi salauskoodeista ja siansaksan lähettämisestä. Sähkeet laskutettiin sanamäärän perusteella, ja toiminnan kannattavuus kärsi, koska salauskoodit lyhensivät viestejä huomattavasti. Selkokielisen tekstin lukeminen ja lähettäminen oli myös paljon nopeampaa kuin pikkutarkan merkkisotkun. Asiasta kiisteltiin pitkään palveluntuottajien ja asiakkaiden välillä. Lopulta ITU antoi ukaasin, että salausta sai käyttää mutta koodin tuli koostua luonnollisista sanoista kahdeksalla sallitulla kielellä: englanti, espanja, hollanti, italia, latina, portugali, ranska ja saksa. Kaikilla muilla kielillä lähetettyjä viestejä kohdeltiin koodina ja niistä perittiin korkeampi taksa.

Duunikavereita ja romanssia

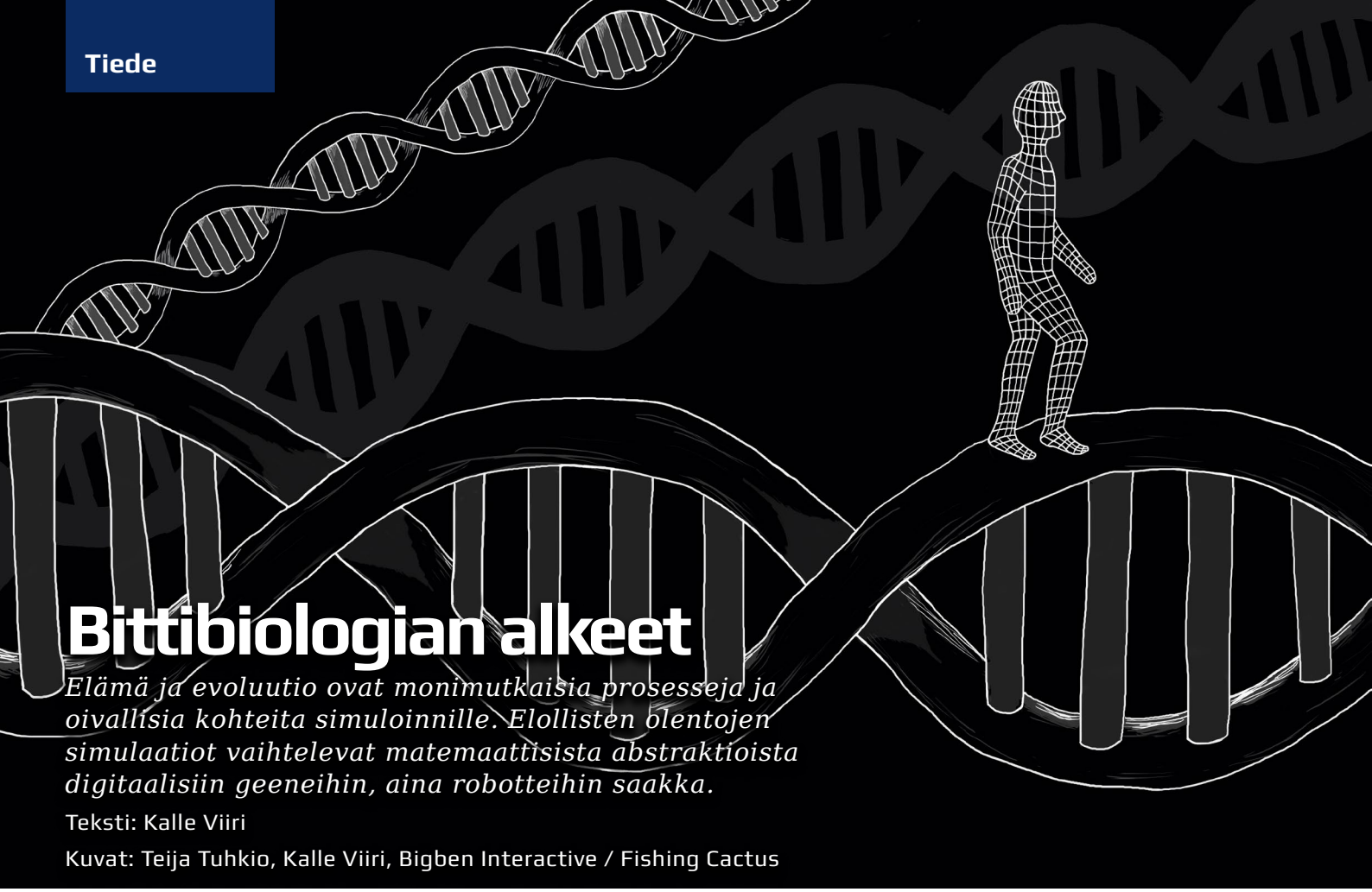
Vaikka sähkösanomat sinänsä ovatkin enimmäkseen verrattavissa nykyajan sähköpostiin, kulki linjoilla paljon muutakin liikennettä. Sähköttäjät kommunikoivat toisilleen reaaliaikaisesti vaihtamalla erilaisia statusviestejä ja arkisia kuulumisiaan. Tämä viestikieli muuttui puolivirallisiksi lyhenteiksi ja jargoniksi hyvin varhain. Päivittäiseen linjan hoitamiseen kuului sellaisia luonnollisia lyhenneilmaisuja kuin ”hyvää huomenta”, ”olen valmis vastaanottamaan” ja ”toista edellinen”.

Lennätinkeskukset oli usein miehitetty varautuen ruuhkahuippuihin, joiden ulkopuolella saattoi olla luppoaikaa. Tällöin sähköttäjät tapasivat viettää aikaa jutellen naapurikeskusten kanssa. Viestintä oli täysin reaaliaikaista vuorottelua linjalla, toisin kuin viralliset asiakkaiden tilaamat sähkeet, jotka kirjattiin paperille sisään tullessa ja vastaanottajalle toimitettaessa.

Sähköttäjillä oli pelkästään lennättimen välityksellä hoidettuja ystävyyssuhteita naapurikaupungeissa tai pidemmälläkin työskenteleviin sähköttäjiin. Järjestelmät toimivat monissa paikoissa pitkään niin, että esimerkiksi New Yorkin ja Bostonin välistä liikennettä hoitivat joka päivä kummassakin päässä samat sähköttäjät. Näiden välille saattoi syntyä vuosikausien työparisuhde, johon mahtui ruuhkahuippujen välillä epävirallista rupattelua ja kuulumisten vaihtoa. Nämä ihmiset eivät välttämättä koskaan tavanneet toisiaan, mutta vuosikausien pituiset ihmissuhteet hoituivat linjoja pitkin aivan samoin kuin nykyäänkin.

Suomessa käytettävät morseaakkoset

A ·−	K −·−	U ··−
B −···	L ·−··	V ···−
C −·−·	M −−	W ·−−
D −··	N −·	X −··−
E ·	O −−−	Y −·−−
F ··−·	P ·−−·	Z −−··
G −−·	Q −−·−	Å ·−−·−
H ····	R ·−·	Ä ·−·−
I ··	S ···	Ö −−−·
J ·−−−	T −	



Bittibiologian alkeet

Elämä ja evoluutio ovat monimutkaisia prosesseja ja oivallisia kohteita simuloinnille. Elollisten olentojen simulaatiot vaihtelevat matemaattisista abstraktioista digitaalisiin geeneihin, aina robotteihin saakka.

Teksti: Kalle Viiri

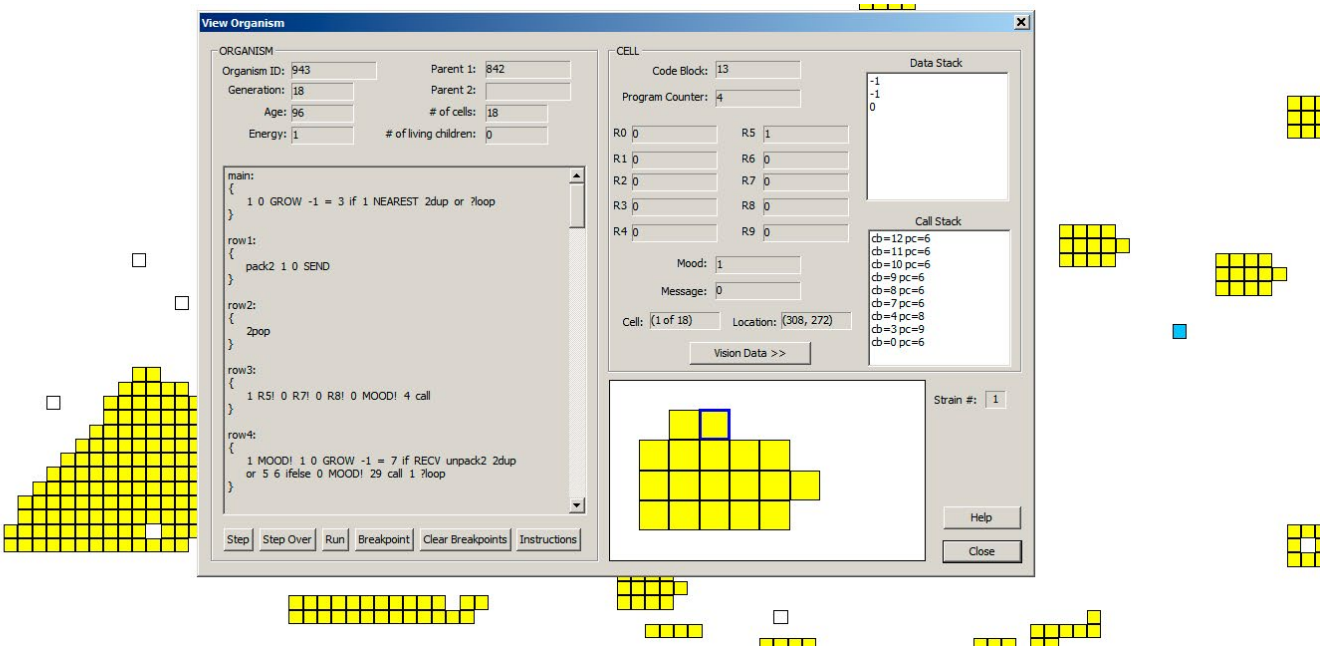
Kuvat: Teija Tuhkio, Kalle Viiri, Bigben Interactive / Fishing Cactus

Soluautomaatio

Matemaatikko John Conwayn kehittämä Game of Life on tunnettu soluautomaatti, matemaattinen ”peli”, jossa solut vaikuttavat toisiinsa ennalta määrättyjen sääntöjen mukaisesti. Solut elävät ruudukossa, ja ne selviävät vain, jos niillä on sopiva määrä eläviä soluja naapureinaan. Alle kaksi naapuria johtaa kuolemaan yksinäisyyden vuoksi ja yli kolme naapuria kuolemaan tungoksen takia. Jos tyhjällä ruudulla on kolme naapuria, siihen syntyy uusi solu.

Yksinkertaisesta säännöstöstä huolimatta Game of Life on täynnä mielenkiintoisia ilmiöitä. Useista lähtötilanteista kehittyy villisti ja pitkään laajenevia kaoottisia solurykelmiä, mutta myös vakaita kuvioita. Vakaita kuvioita kutsutaan oskillaattoreiksi, jos ne liikkuvat, ja still lifeiksi, jos ne ovat täysin paikallaan. Muita mielenkiintoisia kuvioita ovat ruudukolla vaeltavat liitimet (glider) ja avaruusalukset (spaceship).

Game of Lifea pidetään usein keinoelämän alkeellisena muotona, sillä monet solurykelmät käyttäytyvät orgaanisen oloisesti. Pelin mielenkiinto liittyy kuitenkin elämän simuloimisen sijasta pitkälti pelin matemaattisiin ominaisuuksiin. Ohjelmoijat ja matemaatikot ovat haastaneet itseään siten, että etsivät esimerkiksi itseään kopioivia koneita ja tietoko-



Evolve muistuttaa ohjelmointiympäristöä. Tarkasteltavana solun ominaisuudet

neita, jotka voi toteuttaa pelin säännöillä. Suurimmissa tällaisissa rakennelmissa on miljoonittain soluja, ja niiden toiminnasta on erittäin vaikeaa ottaa selvää ihmissilmällä.

Game of Lifen ja muiden soluautomaattien kokeiluun on saatavilla kymmenittäin ohjelmia. Yksi tunnetuimmista lienee GoLLy, josta tämän artikkelin kuvankaappaukset on otettu.

Ohjelmoitavia solurobotteja

Evolve 4.0 muistuttaa ensi vilkaisulla soluautomaattia. Game of Lifen lailla peli keskittyy yksittäisten solujen elämään ja vuorovaikutukseen ruudukkomaailmassa. Ulkonäkö kuitenkin pettää. Evolven solut muistuttavat Conwayn matemaattisen mallin sijasta yllättävän paljon reaalimaailman organismeja.

Jokainen Evolven solu on ohjelmoitu kForthilla, Forth-ohjelmointikielen kaltaisella kielellä, joka määrittää solun toiminnan. Solu seuraa koko elämänsä ajan ohjelmakoodia suorittaen käskyn kerrallaan. Käskyt tekevät kaiken, mikä liittyy solun toimintaan. Ne liikuttavat solua, saavat sen syömään, kasvattamaan uusia soluja osaksi itseään tai tuottamaan itiöitä lisääntyäkseen. Soluilla on myös alkeellisia järjestelmiä toistensa kanssa viestimiseen.

Solut lisääntyvät jättämällä ympäristöönsä itiöitä. Kun kaksi itiötä jätetään samaan ruutuun, niistä syntyy elävä solu. Jos itiöt ovat peräisin samasta eliöstä, kyseessä on suvuton lisääntyminen, muussa tapauksessa suvullinen lisääntyminen. Uusi eliö perii vanhempiensa piirteet ohjelmakoodin muodossa mutta satunnaismutaatioilla höystettynä. Periytyminen toimii melko sujuvasti, koska kForthissa on helposti käsiteltävä rakenne.

Solujen määrää ja niiden käyttämää muistia rajoittaa simulaatiouniversumin energiamäärä, joka pysyy jatkuvasti vakiona. Solu tarvitsee energiaa itiöiden tuottamiseen ja sisäisen muistipinonsa kasvattamiseen. Muut toiminnot ovat solulle ilmaisia, mutta kun sen energia loppuu, se kuolee. Solut palauttavat energiaansa syömällä itiöitä, toisia soluja tai kuolleita soluja, pitäen näin energian kierrossa.

Yksinkertaisuudestaan huolimatta Evolven simulaatiomalli mahdollistaa yllättävän monimutkaisten otusten kehittymisen. Pelin kehittäjä Ken Stauffer on tehnyt useita kokeita, joissa simulaatioiden annetaan toimia todella pitkiä aikoja. Kokeiden tuloksena on syntynyt useasta erikoistuneesta solusta koostuvia organismeja, jotka selviävät merkille pantavan hyvin jopa silloin, kun ne kilpailevat ravinnosta ihmisen ohjelmoimien organismien kanssa.

Norn-lapsi Bob valittelee väsymystään

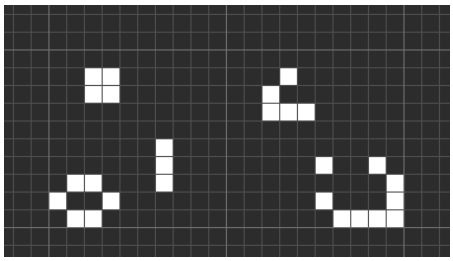
Nornit osaavat kommunikoida keskenään ja pelaajan kanssa yhdistelemällä verbejä ja objekteja.

Kauniit mutta syvälliset

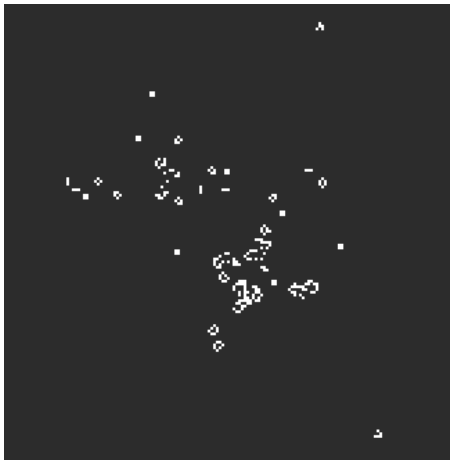
Vuonna 1996 julkaistun Creaturesin ulkokuori on hämäävän suloinen. Päärooleissa olevat norn-hahmot ovat suurisilmäisiä pörröpäitä, jotka kätkevät sisäänsä kiehtovan monimutkaisen simuloidun elimistön, käytösmallin ja perimän.

Nornin elinkaari on hyvin samankaltainen kuin ihmisen. Vastasyntyneet nornit tarvitsevat paljon ravintoa, liikkuvat hitaasti kontaten ja sairastelevat helposti. Ne käyttäytyvät kuten lapset: leikkivät mielellään, kiinnostuvat asioista helposti ja oppivat asioita yrityksen ja erehdyksen kautta. Esimerkiksi liedellä kätensä polttanut norn oppii välttämään kuumia asioita. Pelaaja vaikuttaa nornien maailmaan kädellä, joka toimii kursorina. Kädellä voi siirtää esineitä, käskeä norneja sekä lyödä tai silittää niitä rangaistukseksi tai palkinnoksi.

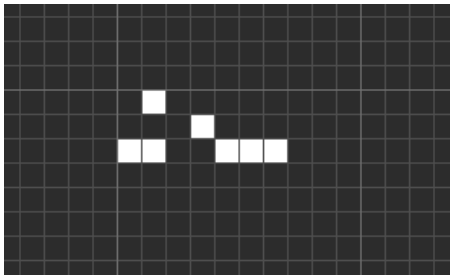
Kasvaessaan norn alkaa liikkua nopeammin ja kiinnostuu toisista norneista enemmän. Olennot osoittavat toisilleen hellyyttä pusuilla ja toisinaan purkavat vihaansa lyömällä toisiaan. Sukupuolihormonien toiminta alkaa aikuisuuden kynnyksellä ja saa nornin kiinnostumaan



Yleisesti nähtyjä hahmoja Game of Lifessa: vasemmalla ”palikka”, ”mehiläispesä” ja ”vilkkuja”, oikealla liidin ja avaruusalus.



Acorn 512 sukupolven jälkeen.



Acorn eli ”käpy”, pieni kuvio, joka laajenee pitkään pelin käynnistymisen jälkeenkin.K

vastakkaisesta sukupuolesta. Sukukypsien nornien välinen suukottelu voi johtaa raskauteen ja jälkikasvun syntymiseen. Lapsinornit perivät vanhempiensa piirteitä pelin perimämallin mukaisesti.

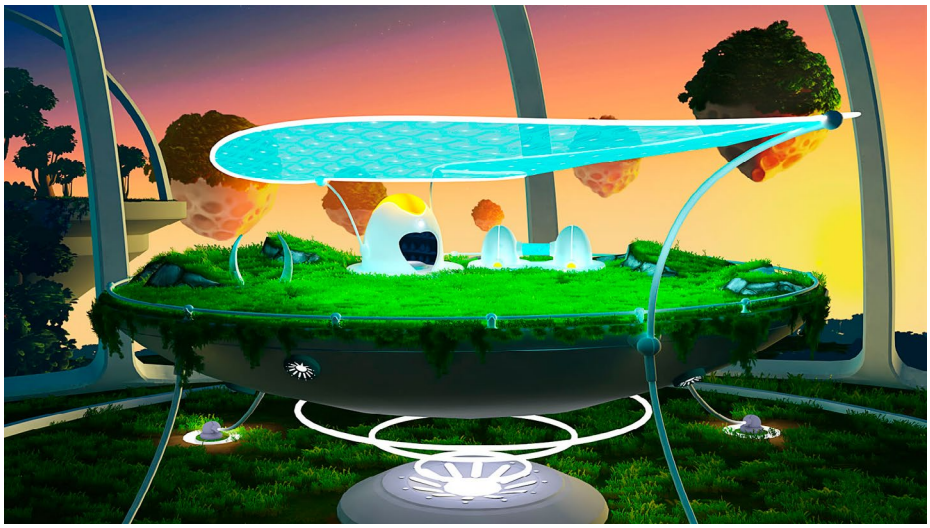
Kun nornit ikääntyvät, niiden hedelmällisyys heikkenee. Niistä tulee hitaampia, ja ne sairastuvat helpommin. Terveyttä voi ylläpitää lääkitsemällä vanhuuden vaivoja erilaisilla kemikaaliruiskeilla, mutta ennemmin tai myöhemmin norn kuolee. Nornin luonnollinen elämä kestää yleensä 5–10 tuntia. Rotu- ja ympäristötekijät vaikuttavat elinikään, mutta norn voi kuolla myös ennen vanhuusikää erilaisiin sairauksiin, myrkkyihin, nälkiintymiseen tai vammoihin.

Pinnan alle

Nornin käyttäytymistä säätelevät kemikaalit, joita on jokaisessa eliössä yli sata erilaista. Suuri osa kemikaaleista kuuluu normaaliin aineenvaihduntaan, esimerkiksi glukoosi, rasva, happi, hiilidioksidi ja C-vitamiini. Kemikaaleja ovat myös myrkyt ja taudinaiheuttajat sekä niiden vasta-aineet. Lisäksi niihin kuuluvat eliön tarpeita kuvaavat vietit kuten sukupuolivietti, kipu ja nälkä. Kemikaalien vaikutukset eivät ole kiinteitä, vaan ne riippuvat paljolti eliön perintötekijöistä.

Kemikaalien eritystä säätelevä geenijärjestelmä on monimutkainen. Se on suunniteltu muistuttamaan todellista DNA:ta rakenteeltaan ja toiminnaltaan. Perintötekijöiden vaihtelu aiheuttaa todella paljon vaihtelua saman lajin yksilöiden välillä. Norneilla voi esiintyä perinnöllisiä ongelmia kuten ruokahaluttomuutta tai heikkoa vastustuskykyä. Niillä voi olla myös toivottavampia piirteitä kuten poikkeuksellisen tehokasta aineenvaihduntaa, harvinaisia värikuvioita, myrkkyimmuniteettia tai jopa kuolemattomuutta.

Risteyttämällä norneja sopivasti pelaaja voi jalostaa haluamiaan piirteitä esiin, mutta se on melko hidasta. Joissakin sarjan peleissä toimintaa voi nopeuttaa käyttämällä ”geeniyhdistäjää”, joka surmaa kaksi olentoa ja muodostaa niiden geeneistä uuden yksilön. Toisena vaihtoehtona on käyttää pelin lisäosana saatavaa geenieditoria ja muokata geenikoodia suoraan.



Norneilla on vähemmän tunnettu mutta merkittävä rooli digitaalisten eliöiden oikeuksiin liittyvässä keskustelussa. 1990-luvun lopulla Creatures-sarjan fanien joukkoon syntyi alakulttuuri, jossa kehiteltiin kekseliäitä tapoja kiduttaa tai surmata norneja. Ilmiön huippuvuosina jaettiin internetissä muun muassa pelokkaaksi mutta itsetuhoiseksi koulutettua nornia, jonka perimään oli peukaloitu kroonisen humalatilan aiheuttava geeni. Monet nornien ystävät suivaantuivat, ja keskustelualueilla otettiin vilkkaasti kantaa siihen, onko digitaalisten eläinten kiduttaminen oikein. Keskustelu käytiin jo muutamaa vuotta ennen kuin perheiden polttaminen elävältä The Simsissä muodostui kansanhuviksi!

Sarjan päävisionääri, englantilainen tietojenkäsittelytieteilijä Steve Grand on tehnyt Creaturesin jälkeen monia mielenkiintoisia keinoelämäkokeita. Yksi kunnianhimoisimmista hankkeista on nytkähdellen liikkuva suurisilmäinen robottioranki Lucy, joka tunnustelee ja katselee sille annettuja esineitä kuin pieni lapsi. Sen toimintaa ohjailee keinotekoinen neuroverkko, jonka avulla Grand toivoo Lucyn oppivan maailmasta omatoimisesti. Lucy toimii osoituksena siitä, että keinoelämää voi esiintyä myös virtuaalimaailman ulkopuolella.

Open Worm

Keinoelämän luominen voi olla tieteen työkalu siinä missä muutkin simulaatiot. Yksi kiehtova esimerkki simuloitujen organismien mahdollisuuksista on OpenWorm -projekti. Avointa lähdekoodia ja dokumentaatiota korostava OpenWorm pyrkii simuloimaan C. elegans -sukkulamatoa yksittäisten solujen tarkkuudella. Sukkulamato valittiin simuloinnin kohteeksi yksinkertaisen hermostonsa vuoksi, ja se niitti tiedeyhteisössä mainetta jo 1998 olemalla ensimmäinen monisoluinen eliö, jonka perimä sekvensoitiin kokonaan.

OpenWorm on vielä keskeneräinen, mutta osittain toimivia solu- ja hermostomalleja pääsee lataamaan ja tutkimaan osoitteessa http://www.openworm.org/.

Skrolli

Peruuttamaton poikkeus tilauksesi jatkuvuudessa. Kaikki tilaukset päättyvät vuoden 2013 loppuun. Ellet uusi tilaustasi, menetät tallentamattomat tietosi ja mahdollisuuden oppia uusia asioita!

* Siirry osoitteeseen skrolli.fi/tilaa
* Jatka lukukokemusta ensi vuodelle syöttämällä osoitetietosi ja painamalla tilausnappia. Menetät 30 euroa, mutta saat neljä laadukasta lehteä kotiin kannettuna.

Paina mitä tahansa näppäintä jatkaaksesi
suomalaisen tietokonekulttuurin tukemista!

Freeciv Longturn

Pitkää vuoroa pukkaa

Kiinnostaako kuukausien pituinen Civ-matsi? Meitä kiinnosti. Tuloksena syntyi yksi tarina, yksi kokemuskertomus ja lyhyt oppimäärä perustietoa kätevässä paketissa.

Teksti: Jaakko Heusala, Ville-Matias Heikkilä, Jarto Nieminen, Tapio Berschewsky
Kuvat: Ricardo Liberato, Ville-Matias Heikkilä, Jaakko Heusala

Freeciv on Civilization-pelisarjan ensimmäisten osien tyylinen vuoropohjainen strategiapeli, joka perustuu kokonaan vapaaseen lähdekoodiin ja on kehittynyt alkuperäisistä Civilization-peleistä eteenpäin.

Freecivin merkittävimpiä eroja Civilization-pelisarjaan on sen lähes loputon muokattavuus. Jos haluaa tehdä pelistä mielenkiintoisemman tai korjata epämiellyttäviä ominaisuuksia, voi muutokset yleensä tehdä helposti palvelimen asetuksia muuttamalla. Vielä enemmän saa aikaan, jos osaa C- tai LUA-ohjelmointikieliä.

Freeciv on viimeisten vuosien aikana kulkenut ripeää vauhtia eteenpäin. Kehitystä on tapahtunut niin yksinpelissä kuin moninpelissäkin. Monipuoliset diplomaattiset sopimukset onnistuvat sekä tekoälyn että toisten pelaajien kanssa.

Jättimäisiä pelejä

Freeciv tukee nykyään jopa 126 pelaajan pelejä ja kahden miljoonan ruudun kokoisia karttoja. Suurten pelien kanssa on tosin suurempi riski törmätä uusiin bugeihin. Harvempi tarttuukin tällaiseen järkälepeliin, joka myös vaatii enemmän tehoja palvelimelta. Merkittävimpiä jättipelejä ovat Longturnit.

Longturnissa Freeciviä tai Civilizationia pelataan yksi vuoro tietyn pitkän ajanjakson kuluessa, kuten kerran päivässä. Kansainvälisessä Longturn.orgissa vuoron pituus on yleensä 23 tuntia, ettei kukaan pelaajista saa etua aikavyöhykkeen ansiosta.

Pelaamiseen riittää periaatteessa varttitunti päivässä. Loppuvaiheessa peliä siihen voi kulua tuntejakin. Erityisesti alussa aikaa menee pelipalvelimen erityispiirteiden opiskeluun. Erot voivat olla hyvinkin suuria riippumatta siitä, onko pelannut aikaisemmin, sillä asetukset vaihtelevat paljon palvelinten välillä. Pelaajat pääsevät yleensä itse vaikuttamaan näihin ennen pelin alkamista äänestyksillä ja esittämällä omia ideoitaan.

Kotimaassa ja muualla

Suomessa longturn-pelejä on järjestetty jo yli kymmenen vuotta. Aluksi pelejä pelattiin muutamassa eri porukassa, jotka ovat sittemmin yhdistyneet Freeciv.fin alle. Verkkosivujen lisäksi yhteisön löytää Ircnetistä kanavalta #freeciv.fi.

Freeciv.fi aloitti maailman ensimmäisen 126 ihmispelaajan longturn-pelin syyskuussa 2011, noin kuukausi ensimmäisen tähän kykenevän Freecivin version 2.3.0 julkaisun jälkeen.

Maailmalla on nykyään myös kaksi kansainvälistä peliyhteisöä, Longturn ja Greatturn. Näihin peleihin osallistuu yleensä noin 40-60 pelaajaa per peli.

Pelaaminen näissä yhteisöissä vaihtelee paljon ja säännöt vaihtelevat, mutta yhteistä kaikille on, että pelaajat päättävät asioista yhdessä. Peliä myös kehitetään porukalla paremmaksi: tuodaan omia ideoita esiin, keskustellaan niistä ja tehdään niistä yhdessä todellisuutta.

Jaakko Heusala

Jumalaisen komea Wenden, vain hieman alun jälkeen. Teutonisen ritarikunnan maailmanvalloitussuunnitelmat näyttivät tässä vaiheessa vielä hyviltä. Miten pian ne murskaantuisivatkaan.

Miksteekkien alkukoti loistaa varhaisessa kukoistuksessaan niin puhtaana, kauniina ja yksimielisenä. Vuorolla 14 uskottiin vielä, ettei merten takana olisi mitään tai ketään.

Diplomaattilaiva on juuri saapunut Sveitsin pääsaaren rantaan. Sveitsin valtionjohto oli kuitenkin vetäytynyt jo ennen kontaktia.

Ensi kertaa pitkässä pelissä

Loppuvuonna 2012 alkanut Longturn-peli numero 31 oli ensimmäinen Freeciv-moninpeli, johon osallistuin yli kymmeneen vuoteen. Olin havainnut aikoinaan moninpelit liian hankaliksi ajankäytön kannalta, mutta Longturnista kuuleminen palautti kiinnostukseni. Peli kestää kuukausitolkulla, mutta koska sitä pelataan vain vuoro päivässä, pitäisi sen ainakin periaatteessa sopia paremmin kalenteriin.

Peli 31 käytiin saarikartalla, jolta kukin pelaaja sai alussa täysin oman mantereen. Alkupeli olisi tästä syystä varsin yksinäistä puurtamista. Suunnittelin kehityslinjan hyvin tarkkaan ajatuksenani keskittyä etenkin rauhanomaiseen tieteeseen. Tein saaresta itselleni kopion, jolla simuloin läpi mahdolliset kehityskulut kymmeniä vuoroja eteenpäin: Millä ratkaisuilla saisin kirjastot kaupunkeihin mahdollisimman nopeasti? Pitäisikö tähdätä ensin monarkiaan vai yliopistoon? Vaikka periaatteessa vuorot saattoi hoitaa muutamassa minuutissa, käytin välillä simulointiin ja taustatarinointiin jopa tuntikausia.

Yksinäisyyden aikaa kesti lopulta yli kaksi kuukautta. Kaikki pikkuvenein saavutetut saaret olivat asumattomia, ja vasta syvän meren laivat alkoivat löytää naapureita. Ensimmäinen naapurini, Sveitsi, oli vajonnut koomaan juuri paria päivää ennen ensiyhteyttä, kun pelaaja oli kyllästynyt yksinäisyyteensä. Oikea kontakti saatiin vihdoin vuorolla 73 itävaltalaisiin ja heidän Raar-liittoumaansa, ja liityin melko pian Raarin kolmanneksi jäseneksi.

Rajat murtuvat

Ensikontakti muutti pelin luonteen täysin. Miksteekkien alue oli vain pieni saarirykelmä valtameressä, jota kansoitti yli 70 sivilisaatiota. Kansat muodostivat liittoumia, joiden kesken vaihdeltiin karttoja ja haalittiin uusia jäseniä. Yhteyttä pidettiin ircissä, jabberissa ja keskustelufoorumeilla. Pian ei enää riittänyt, että teki siirtonsa jossain vaiheessa vuorokautta, sillä etenkin sotaoperaatioissa myös vuorokaudenaika oli tärkeässä osassa.

Itselleni dramaattisin käänne tapahtui vuorolla 148, kun itänaapurin pääsaari vallattiin kokonaan yhden ainoan vuoron aikana. Tällöin rauhanomaisenkin sivilisaation piti ruveta varustautumaan kunnolla. Maailma polarisoitui: sotahullujen ekspansionistien yhdistäessä voimiaan Raar, Kirkko ja muutamat muut rauhanomaiset liittoumat muodostivat oman yhteenliittymänsä. Voimatasapaino muuttui etenkin Kirkon jäsenten hyvän koordinointityön ansiosta, ja rauhan viholliset kukistettiin lähes täysin ennen vuoroa 180.

Koska Longturn-otteluita on pelattu jo kymmenen vuotta, ajattelin pelitapojen hioutuneen sen verran, että tulokkaan saattaisi olla vaikea pärjätä. Kävi kuitenkin ilmi, etteivät monet kokeneetkaan pelaajat olleet jaksaneet paneutua sääntöihin ja asetuksiin kunnolla, vaan sortuivat perustavanlaatuisiin virheisiin. Odotin joutuvani imperialistien jyräämäksi melko varhaisessa vaiheessa, mutta selvisinkin kunnialla loppuun asti.

Osa luhistuu tielle

Aiemmissa otteluissa ongelmallisiksi olivat kuulemma muodostuneet teknologianvaihtoringit, joita rakentamalla etenkin isot liittoumat pystyivät kehittymään järkyttävän nopeasti. Teknologian siirtämistä oli tästä syystä vaikeutettu, ja mukana oli jopa riski, että vaihdon molemmat osapuolet unohtavat tietämyksensä. Vaihtoja ei siis pahemmin edes yritetty, vaan hitaammin kehittyneille liittolaisille pyrittiin järjestämään tilaisuuksia käydä teknologiavarkaissa isojen vihollisten mailla.

Sääntöihin liittyvät erimielisyydet aiheuttivat myös skismaa. Erään pitemmän palvelinongelman aikana nettiin ilmestyikin kilpaileva Greatturn, jossa pelaajat äänestivät käyttöön otettavista säännöistä enemmistödiktatuurin keinoin. Osallistuin myös hyvin ahtaalla yhtenäismantereella käytyyn Greatturniin, enkä pärjännyt siinä läheskään yhtä hyvin kuin Longturnissa.

Pelin kesto johti myös siihen, että monet pelaajat rupesivat ennen pitkää idlaamaan, eli kyllästyivät peliin antamatta sitä kaverinsa jatkettavaksi. Idlaajasaarten valtaaminen katsotaan Longturn-kulttuurissa täysin hyväksytyksi toimenpiteeksi myös rauhanomaisen linjan vetäjille.

Kokeilun arvoinen

Vaikka Longturn saattaakin ajankäytöllisesti muistuttaa kasuaalipelejä, ei se missään nimessä ole sellainen. Kuukausien päivittäinen sitoutuminen on tärkeää, kuten myös hyvä yleinen ote Civilizationiin. Päätin itse olla osallistumatta peliin 32, sillä se ajoittui kesälle, jolloin päivärutiinista kiinni pitäminen on muutenkin hankalampaa.

Muutamista ongelmista huolimatta suosittelen Longturnin tai Greatturnin kokeilemista kaikille Civilizationeista pitäville – etenkin niille, joilla on pitkäjänteisyyttä mutta ei aikaa tuntikausien pelisessioille.

Ville-Matias Heikkilä

180 vuoroa longturn-freeciviä

Tämä historiallinen kertomus perustuu Teutoniritarikunnan Suurmestari Metonin saneltuihin muistiinpanoihin. Osa tarinasta on kirjuri Thihin tulkintaa, osa suoria otteita Suurmestarilta ja osa ehkä pelkkää legendaa...

Kaikki alkaa vuonna 4000 eaa. Teutoniritarikunta on juuri perustanut ensimmäisen asuinpaikkansa, pääkaupunki Wenden. Hyvin nopeasti siirtokunta kasvaa kahdella lisäkylällä. Königsberg ja Marienburg ovat vaatimattomia valtakunnan ylpeyden rinnalla, mutta silti merkittäviä agraarisia keskuksia.

Suurmestari päättää viisaudessaan, että kolmesta uudisraivaajaryhmästä yksi lisättäisiin pääkaupungin asukkaisiin. Jotain kuitenkin menee vikaan viestiketjussa, ja paikallinen vasallihallitsija päättää käyttää uudisraivaajia orjatyövoimana pääkaupungin tuotannon lisäämiseksi. Kymmenen tuhatta kansalaista kuolee Wendenin kaduille loppuun piiskattuina.

Tämä on vasta alkusoittoa Suurmestarin vihan liekeille. Hänen uransa tulee olemaan täynnä tuskaa, vääriä päätöksiä, selkään puukottavia liittolaisia ja sen myötä suoranaista pelkuruutta.

Aikansa saartaan tutkittuaan, Suurmestari huomaa harmikseen, ettei hän hallinnoi kovinkaan suurta mannerta. Tiedustelijat näkevät kuitenkin läheisen matalikon yli toiselle maalle. Tämä saa

Kun miksteekit liittyivät raramurien ja itävaltalaisten perustamaan Raar-liittoon, alkoivat kartan mustat alueet täyttyä vauhdilla. Maailman loputtomat rikkaudet odottavat ottamistaan yhdistyneen mahdin edessä.

Suurmestarin määräämään valtakunnan viisaat kehittämään entistäkin tehokkaampia aseita ja varustautumaan mahdolliseen miehitykseen.

Suhteet ympäröiviin valtoihin eivät kuitenkaan ole välittömästi yhtä kylmät tai jyrkät kuin Suurmestari alun perin oletti.

Ensimmäinen kosketus

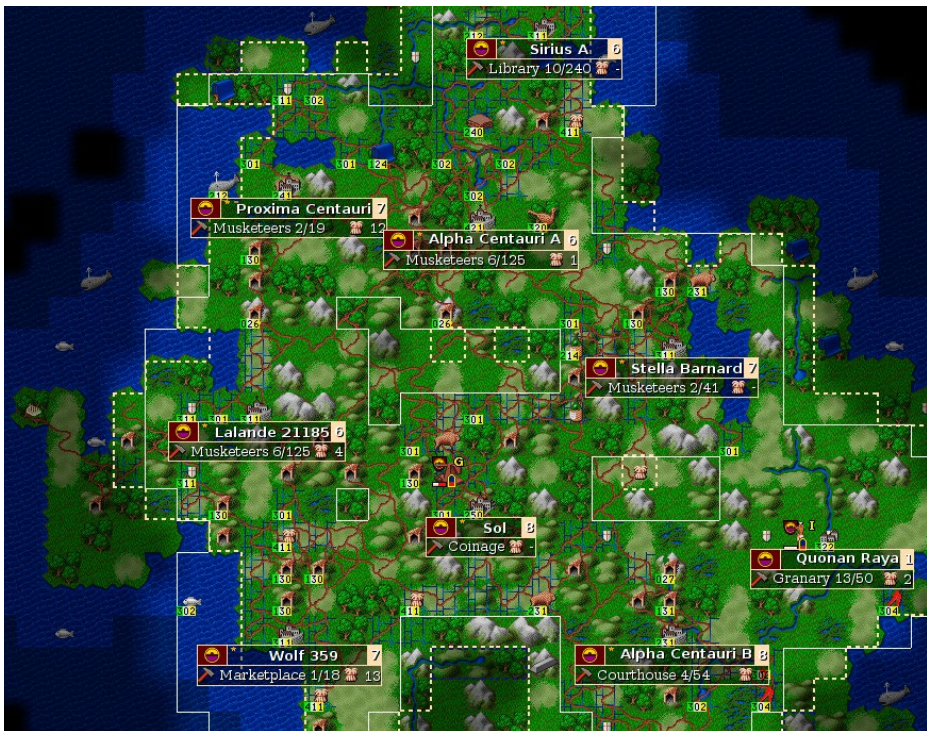
Hyvin nopeasti muut valtiot alkavat liittoutua isommiksi kokonaisuuksiksi. Luonnollisesti kristillisenä valtiona Teutoniritarikunnan Suurmestaria kiinnostaa jäsenyys rauhan filosofiaa edustavassa Kirkon liitossa. Siihen liittyminen tuntuu hieman paradoksaaliselta sotilaalliseen varusteluun panostaneelle Suurmestarille. Kirkko on toisaalta suurin liitto, ja sen jäseniin on valmiiksi diplomaattiset suhteet.

Suhteellisen läheltä löytyy myös Raar-liitto (Ramaruni-Austria), johon kuuluu muutama lähemmin historiasta tunnettu kansa. Liittoneuvottelut käyvät hetken aikaa kuumina, ja Suurmestari onnistuu saavuttamaan alustavan paikan liiton jäsenehdokkaana. Raar-liittouman kartta jaetaan Suurmestarille.

Suurmestari olettaa tilanteestaan liikoja ja luulee tulleensa osaksi Raar-liittoa. Valtakunnan tuotanto suunnitellaan uudelleen. Veneiden rakentaminen jää kymmeneksi vuoroksi sivuun Suurmestarin pyrkiessä tyydyttämään Raar-liiton maanviljelytarpeita.

Kymmenen vuoron päästä Suurmestari saa tietää, että Raar-liitto on päättänyt hylätä Teutoniritarikunnan. Kymmenen tuhannen kansalaisen menetys jo ensimmäisellä vuorolla on hidastanut ritarikunnan kehittymistä. Raar-liitto on tämän vuoksi päättänyt, etteivät Teutonit ole tarpeeksi merkittävä ja suuri valtio liittolaisekseen.

Suurmestari vannoo kostoa Viznutille, Bamskampille ja kaikille muille tyhjiä lupauksia tehneille Raarin johtajille. Kirotut petturit saisivat vielä maistaa oikeuden kylmää terää. Veneiden kehitys palautetaan kansakunnan ykköspriooriteetiksi välittömästi. Veneillä sanan-



Sveitsin tyhjiksi jääneet temppelit ja autioituneet kadut oli helppo vallata. Miksteekit eivät kohdanneet vastustusta siirtäessään omat nukkehallitsijansa paikalle. Kukaan ei edes älähtänyt, kun salkoon vedettiinkin valkoristin sijaan kaunis maanläheinen miksteekkilippu.

saattajat raivaavat ensitöikseen tiensä Kirkon liiton jäsenten pakeille. Liittosopimus syntyykin nopeasti. Neuvotteluissa Raar-liitolta saadut kartat ja tiedustelutiedot ovat kelpoa kauppatavaraa.

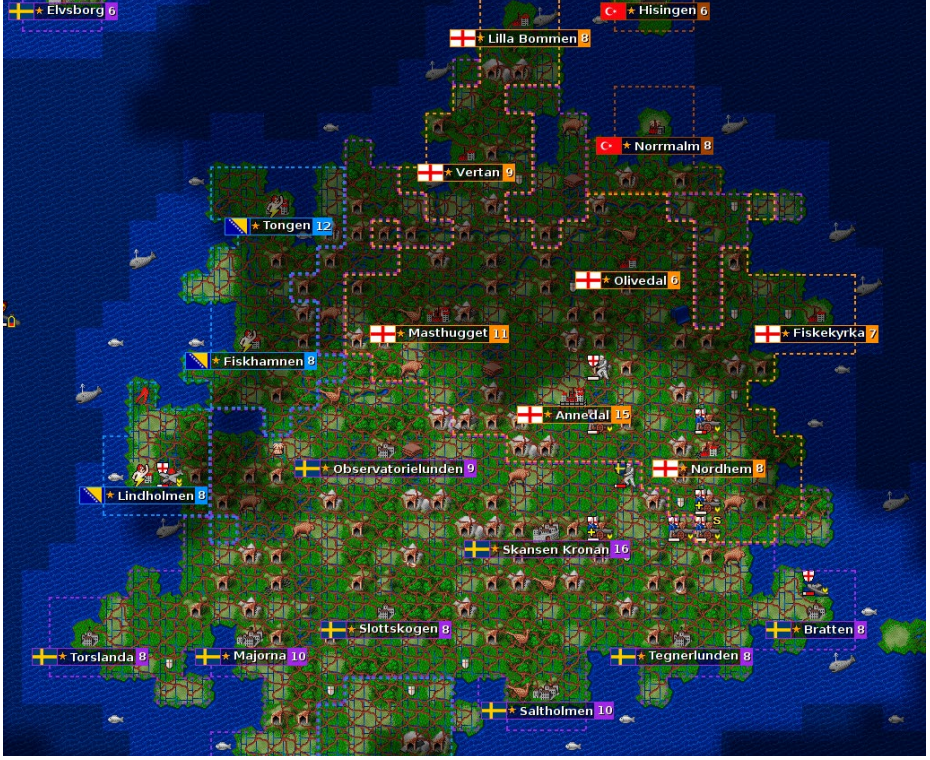
Asiat kehittyvät kuitenkin nopeasti ja odottamattomin tavoin. Pian sekä Raar että pari muuta ympäröivää liittoa kuuluvat Kirkkoon. Kostosuunnitelmat ja vanhat kalavelat unohtuvat suuremman kuvan avautuessa Suurmestarille.

Uskonsodan alkumetrit

Yhteistyö kirkon kanssa asettuu kymmeniksi vuoroiksi tasaiseksi ja jopa tylsäksi maanviljelyksi. Ensimmäisten vuorojen virheiden päälle kerääntyy nippu uusia takaiskuja. Malttamaton Suurmestari ei ole tyytyväinen peltojen raivaamiseen, vaan kaipaa voitokkaaseen sotaan.

Vihdoin loppupeli koittaa, ja Suurmestarin ensimmäinen sotaretki alkaa. Kirkon liiton sotilasmestari Xercise lähettää Teutoniritarikunnan varustamolle tilauksen. Suureen sotaan kaivataan joka mies, vaikka Suurmestarin epäonni ja taidottomuus tieteen ja talouden hoidossa tarkoittaakin sitä, että hänen joukkonsa ovat teknologisesti alivarusteltuja viholliseen verrattuna.

Tilaukseen kuuluu 3 kaljuunalastillista parhaita musketöörejä ja tykkejä. Yhteen kaljuunaan mahtuu 4 komppaniaa miehiä. Yhteensä tilauslistalla on siis 12 komppanian armeija Teutoniritareiden tykistöä ja musketöörejä. Teutoniarmeijan upseerit ja kouluttajat ryhtyvät välittömästi toimiin vätysten piiskaamiseksi ruotuun.



Turkkilaiset ja englantilaiset ovat juuri tunkeutuneet Ruotsin pääsaarelle. Saari vallattiin kokonaan samalla vuorolla, mikä jätti ruotsalaisille vain muutaman syrjäisen pikkusaaren.

Hyökkäysjoukot valmistuvat teutonisella tehokkuudella etuajassa ja lähtevät pitkälle purjehdukselle kohti liittolaisvaltio Itä-Saksan rannikkoa. Armeijan on tarkoitus nousta maihin poukamassa, joka on sodan etulinjassa. Täällä joukkojen on määrä odottaa Kirkon yhteistä maihinnousuhyökkäystä pahuuden voimia vastaan.

Kunniaton tappio

Joukot saapuvat perille turvallisesti ja ajoissa. Suurmestarin päiväkäsky joukkojen rantauttamisesta ei kuitenkaan löydä tietään Amiraalin toimistoon. Teutonien eliitit jäävät siis odottamaan laivoihin rannikon tuntumaan – valmiina hyökkäykseen. Kaljuuna-armadaa suojelee vastarannalta lähestyviltä vihollisilta Hakkereiden, Itä-Saksan ja Ranskan hävittäjälaivastot.

Salakavala vihollinen on kuitenkin kaikkia ovelampi ja hiipii laivaston selustaan sukellusveneillään. Syvyyksistä ilmaantuvat tuholaiset upottavat vaivatta koko Teutoniritarikunnan vanhentuneen kaluston. Myös iso osa liittolaisten moderneista hävittäjistä uppoaa samassa rytäkässä.

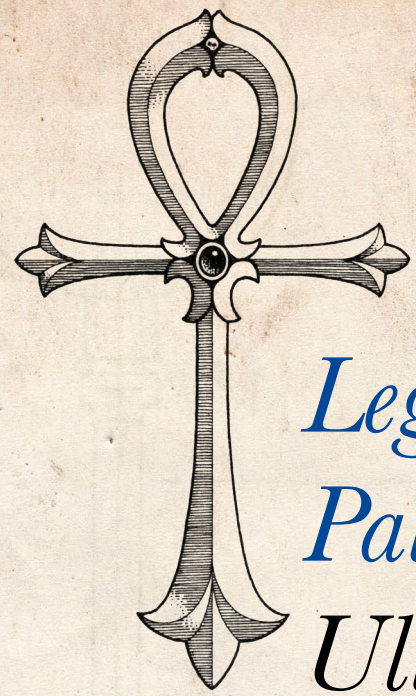
Murskaava tappio käy raskaaksi Suurmestarin moraalille. Syvä masennus ajaa hänet etsimään neuvoa pullon pohjalta. Loput hallintokaudestaan Meton viettää enemmän tai vähemmän päihtyneenä kiroten kurjaa kohtaloaan.

Kaikki yritykset auttaa tai tukea Kirkon eteneviä joukkoja valuvat hiekkaan. Laivaston antiikkinen purjekalusto ei yksinkertaisesti ole tarpeeksi nopea pysyäkseen modernimpien laivojen perässä. Aivan viimeisillä vuoroilla saavutetaan pieniä, mutta kokonaiskuvassa merkityksettömiä voittoja Palatinaattiritarikuntaa vastaan Syyrialaisten armollisella avustuksella.

Planetfall jää Teutonien ainoaksi sotasaaliiksi, ja Suurmestari vaipuu lopulliseen epätoivoon. Rauhanliiton maanviljelijöiden marssiessa valtaamaan maailmaa hän vaipuu hiljaa pois yksinäiseen tuntemattomuuteen. Teutoniritarikunnan taival päättyy hiljaiseen ja hampaattomaan raivoon.

Jarto Nieminen,
Tapio Berschewsky

Viimeisellä vuorolla liittoumat tekevät kaikkensa, jotta saisivat vallattua vielä muutaman takapajuisen kaupungin. Tankit vyöryvät eteenpäin viimeiseen saakka, ja ydinpommit paukkuvat.



Legendojen Paluu: Ultiman ja Wastelandin Perilliset

Joukkorahoituspalvelu Kickstarterin menestystarinat ovat innoittaneet haudanryöstäjiä etsimään muinaisia peliaarteita. Kaksi joukkorahoituksen avulla uutta tulemistaan tekevää peliä on yli muiden.

Teksti: Jukka O. Kauppinen
Kuvat: Jukka O. Kauppinen, Risto Mäki-Petäys, Mobygames

Roolipelaajille on luvassa hieno pelivuosi, joka lupaa palauttaa veteraanikörmyt aikaan, jolloin kaikki oli paremmin. Genren oikeutettu jumalhahmo Richard ”Lord British” Garriott palaa pelikentälle fantasiaroolipelillä Shroud of the Avatar: Forsaken Virtues. Shroud ei ole Ultima, mutta se on temaattisesti selvä paluu Ultima-saagan varhaisempiin vuosiin.

Roolipelilegenda Brian Fargon unelmaprojekti ei sekään vaikuta hassummalta. Wasteland 2 on toistakymmentä vuotta suunniteltu jatko-osa kaikkien post-apokalyptisten seikkailujen isälle.

Kummastakaan pelistä ei voi kiittää perinteisiä julkaisijoita. Ennen niin kunniakas Ultima-sarja on denegeroitunut mikromaksuseikkailuiksi, ja Wastelandin jatko-osan suunnitelmat kustantajat hylkäsivät kerta toisensa jälkeen.

Laaja ja uskollinen faniyhteisö intoutui kantamaan klassikoiden jatko-osille muhkeat rahoituspotit Kickstarterin kautta. Shroud of the Avatar haki miljoonan dollarin rahoitusta, mutta kassaan on kirjoitushetkellä kertynyt jo 2,7 miljoonaa, josta 1,9 Kickstarterista. Wasteland 2 haki 900 000 dollaria ja sai 2,9 miljoonaa.

Keräysten suosio osoittaa, että pelaajakansa on enemmän kuin halukasta näkemään vanhoille klassikoille perinteitä kunnioittavia jatko-osia, joita tehdessä ei tarvitse nöyristellä kaupallisuuden paineiden alla.

Miksi muinaisten pelisarjojen joukkorahoitukseen osallistutaan näin innolla? Asiaa täytyy tutkia aikakoneen kautta. Onneksi Skrollilla on sellainen!

Reikänauhaseikkailuja 70-luvulla

Eletään vuotta 1977. Richard Garriott on 15-vuotias lukiolainen, joka tutustuu koulussa uuteen ilmiöön: tietokoneisiin. Niille ei ole vielä edes kunnon opetusohjelmaa, joten Garriott ylipuhuu rehtorinsa antamaan luvan itse suunnitellun ja itseohjatun tietokoneohjelmointikurssin suorittamiseen.

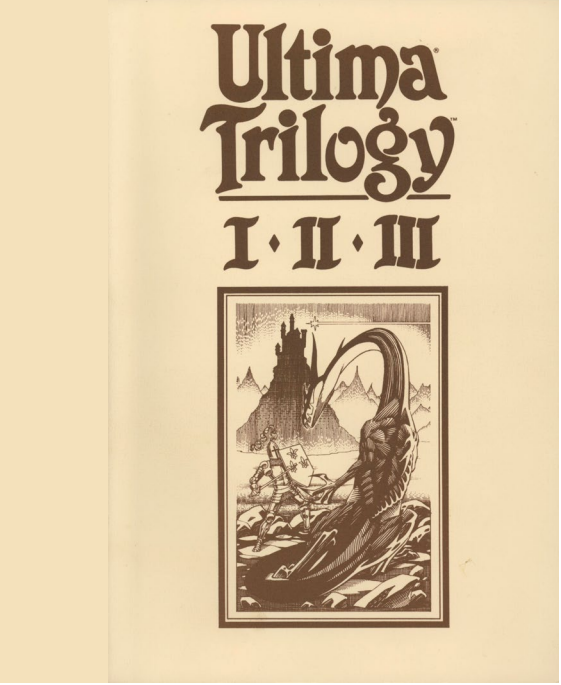
Seuraavan kahden vuoden aikana poika kirjoittaa koulussa kaukokirjoitinpohjaiselle tietokonejärjestelmälle hurjat 28 tekstipohjaista fantasiaseikkailupeliä, läpäisten tietokonekurssin haasteet täysin pistein. Tietokonepohjaisen seikkailemisen lumosta voi saada etäisen mielikuvan kun muistaa, että kaukokirjoitin oli kuin kirjoituskone, johon kuului reikänauhalukija. Koneessa ei ollut lainkaan magneettista tallennusmediaa.

Kesällä 1977 Richard vietti seitsemän viikkoa Oklahoman yliopiston tietokoneleirillä, jossa hän sai lempinimen. Tuosta kesästä lähtien Richard oli Lord British.

Vuonna 1979 nuorukainen oli töissä

Ultimoiden historia

Akalabeth: World of Doom (1980)
Ultima I: The First Age of Darkness (1981)
Ultima II: The Revenge of the Enchantress (1982)
Ultima III: Exodus (1983)
Ultima: Escape from Mt. Drash (1983)
Ultima IV: Quest of the Avatar (1985)
Ultima V: Warriors of Destiny (1988)
Ultima VI: The False Prophet (1990)
Worlds of Ultima: The Savage Empire (1990)
Ultima: Worlds of Adventure 2: Martian Dreams (1991)
Ultima Underworld: The Stygian Abyss (1992)
Ultima VII: The Black Gate (1992)
Ultima Underworld II: Labyrinth of Worlds (1993)
Ultima VII Part Two: Serpent Isle (1993)
Ultima VIII: Pagan (1994)
Ultima Online (1997
Ultima IX: Ascension (1999)
Lord of Ultima (2010)
Ultima Forever (2013)



teksasilaisessa tietokonekaupassa, jossa hän näki ensimmäisen kerran Apple II -tietokoneen. Siinä oli oikeaa grafiikkaa, värejä ja jopa äänet! Richard intoutui tekemään Applelleen ensimmäisen kunnollisen roolipelinsä, Akalabethin. Se teki kaupan omistajaan niin suuren vaikutuksen, että tämä ylipuhui Richardin tekemään siitä myös kaupassa myytävän version.

Akalabethia myytiin tusinan verran, ja yksi kappale päätyi kalifornialaiselle pelijulkaisijalle. Tämä teki pikavauhtia julkaisusopimuksen, jota seurasi yllätysmenestys: Akalabeth myi nopeassa tahdissa 30 000 kappaletta. 17-vuotias Richard tienasi huvikseen tekemällään pelillä ja 200 dollarin panostuksella 150 000 dollaria (nykyrahassa 555 000 dollaria), ja yliopisto-opintojen rahoitus ratkesi kerralla.

Pimeyden ensimmäinen aika

1981 oli merkittävä vuosi. Silloin ilmestyivät Indiana Jones ja kadonneen aarteen metsästäjät, Das Boot ja Mad Max 2. Niin myös Frogger, Haunted House, Jupiter Lander, Pac-Man ja Softporn Adventure. Ja Ultima I.

Garriottin ensimmäinen kaupalliseksi suunnittelema seikkailu viitoitti hänen näkemyksiään siitä, millaisia tietokonepelit voisivat olla. Ultima I olikin todellinen mestarinäyte. Vaikka se on nykysilmin katsottuna kömpelö ja vaivalloinen, oli pelissä selvästi tunnistettavia teemoja. Graafinen tiilityyli pysyi lähes samana aina Ultima V:een saakka. Keskustelujärjestelmä oli alkeellinen, mutta esimerkiksi hahmonluonti onnistui jo neljän eri rodun ja ammatin paletilla.

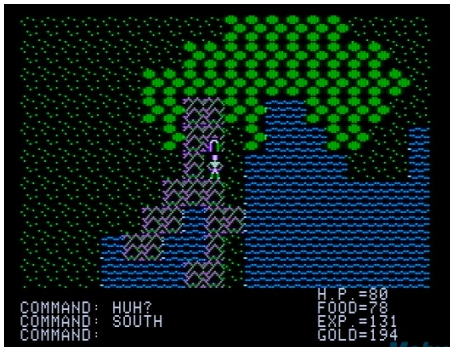
Lisäksi pelaaja oli osa suurta maailmaa, jonka dramaattisessa tarinassa Sosarian maailmaa uhkaa velho Mondain. Sosarian rauhaa rakastava hallitsija Lord British kutsuu avukseen sankarin, joka voi nujertaa Mondainin ja tehdä maailmasta taas hyvän paikan elää. Lord Britishin lisäksi pelissä oli toinenkin persoonallinen henkilö, Iolo the Bard, joka on nähty sittemmin yhtä vaille jokaisessa pääsarjan Ultimassa.

Ultima VII oli isometrisen perspektiivinsä kera jo varsin näyttävä.

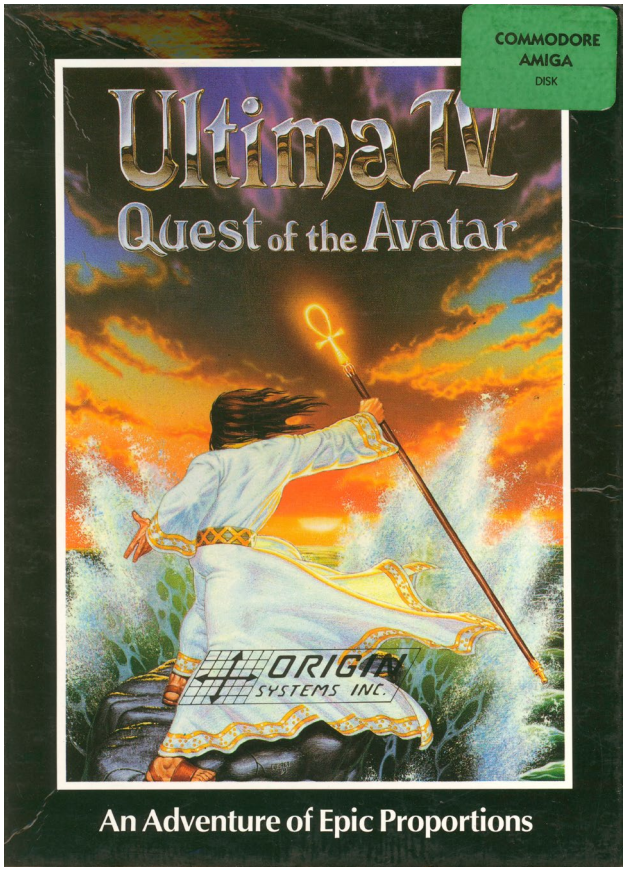
Ultima poiki jatko-osia nopeaan tahtiin. Ultima II: The Revenge of Enchantress (1982) ja Ultima III: Exodus (1983) rakensivat esikoisteoksen hyvis vs. pahikset -teeman päälle, laajentaen Sosarian tarinaa ja vieden sitä kohti uutta, ensimmäisen trilogian katastrofien jälkeistä maailmaa, jonka opimme tuntemaan Britanniana. Sarja lanseerasi nopeaan tahtiin monia nykypelien standardeja. Ultima III esitteli esimerkiksi useasta hahmosta koostuvat seikkailijaryhmät, rajoitetut näkökentät, värikkäät luolastot ja erillisillä kartoilla käytävät taistelut.

Ultima IV vasta olikin kova juttu, ai-

Ultima IV muistetaan erityisesti alun hahmonluonnistaan, joka tapahtui ennustajaeukon kysymyksiin vastaamalla.



Ultima I oli mullistava sukellus tietokoneroolipelaamisen maailmaan.



nakin meille suomalaisille. Siitä suuri kiitos Niko Nirvin MikroBitissä julkaistulle arvostelulle, jonka ansiosta peli breikkasi kovaa. Sosarian raunioille perustettua Britanniaa hallitsi viisaasti ja jalosti oikeamielinen Lord British. Rauha vallitsi, mutta kaivattiin viisautta, esikuvaa – ehkä jopa täydellistä ihmistä.

Ultima IV oli Garriottin omien sanojen mukaan peli, jossa hän opetteli tarinankertojaksi. Se oli myös yksi ensimmäisistä tietokonepeleistä, jossa eettisyydellä ja pelaajan moraalisilla arvoilla oli merkitystä. Hahmo luotiin nopanheiton sijaan ennustajaeukon kysymyksiin vastaamalla, ja pelaajan käytös vaikutti tarinaan hämmästyttävillä tavoilla. Tämän täytyi olla valaistumista ja viisautta etsiessään hyveellinen ja esimerkillinen sankari. Näkökulma oli poikkeuksellinen silloin ja on sitä yhä tänäänkin, kun vastassa ei olekaan suurta myyttistä pahuutta. Pelillä oli myös muita mullistavia ansioita, kuten avainsanoihin perustuva ainutlaatuinen keskustelujärjestelmä, vuorokaudenajat ja maaston vaikutus liikkumiseen. Syystäkin legenda.

Monille sarja huipentui seuraavassa osassa, 1988 ilmestyneessä Ultima V:ssä. Lord British oli kadonnut vasta löydetyn alamaailman uumeniin, ja valtaistuimelle nousi sijaishallitsija, vihattu Blackthorne.

”Mitä Ultima Viidessä ei ole, sitä ei tarvita”, sanottiin. Ehkä näin. Viitosessa peli oli kehitetty täydellisimmilleen, ja puritaanien mukaan sen jälkeen on menty vain alamäkeä.

Uusi aikakausi ja tarinan päätös

Ultima VI: The False Prophetin myötä Ultima-saagan maine oli korkeimmillaan. Pääpelien rinnalle synnytettiin rinnakkaiset pelisarjat Worlds of Ultima ja Ultima Underworld.

Samalla Origin Systems suuntasi rohkeasti kohti kuuttatoista bittiä ja keskittyi tekemään pelejä PC:lle. VGA-värimaailmaan ja isometriseen kuvakulmaan siirtyminen oli joillekin liikaa, mutta Ultimoiden suosio ja mittakaava vain kasvoivat. Nyt koko pelimaailma kuvattiin samassa mittasuhteessa niin maan päällä, kaupungeissa kuin luolastoissakin. Seikkailtavaa riitti.

Kun ensimmäiset kolme Ultimaa kuvasivat Sosarian maailman pimeää aikakautta, osat IV–VI kuvasivat valaistumisen aikaa, jolloin etsittiin puhdasta sielua ja keskityttiin ihmisen sisimmän tutkimiseen. Seuraava neliosainen trilogia syöksyi jälleen synkkyyteen. Etenkin Ultima VIII: Pagan (1994) ja IX: Ascension (1994) kertoivat synkempiä ja pelottavampia tarinoita.

Ultima IX päätyi myös kirjaimelliseen kehityshelvettiin. Viiden vuoden aikana peli tehtiin uusiksi neljä kertaa ja tekijät vaihtuivat tiuhaan tahtiin. Ultima-kaanonia rikkonut seikkailu ei vastannut fanien odotuksia. Kun sekä Ultima VIII että IX myivät odottamattoman kehnosti, pisti Origin Systemsin omistava EA jäihin sekä jatko-osien, lisälevyjen että Ultima Online 2:n kehityksen. The End. Fin.

Joukkorahoitettu lupaus huomisesta

Richard Garriott kätyneineen seurasi Tim Schaferin Double Fine -studion häikäisevää joukkorahoitusmenestystä kiinnostuksella ja totesi, että aika on kypsä selvittää ovatko pelaajat yhä kiinnostuneita Ultima-henkisistä roolipeleistä. Olivat he. Maaliskuussa 2013 avattu Kickstarter-keräys haali puolessa vuorokaudessa 500 000 dollaria.

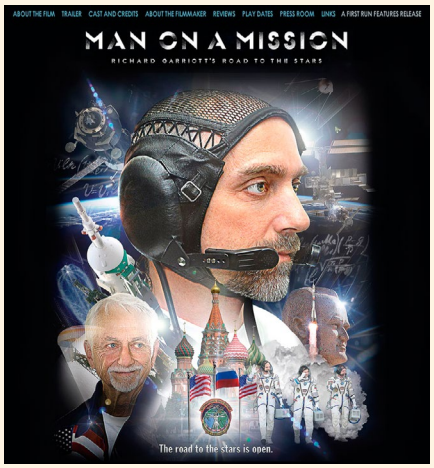
Tuleva roolipeli ei ole Ultima, vaikka se kovin siltä kuulostaa. Jo nimi – Avatarin kääreliinat: hylätyt hyveet – antaa ymmärtää, että Avatar on pistetty hautaan ja aloitettu kaikki alusta. Sattumasta tuskin on kyse, sillä juuri tätä teemaa Ultima-sarjan hengellinen jatko-osa tavoittelee.

Shroud lupaa paluuta vanhan ajan tarinavetoiseen, interaktiiviseen ja moraaliseen roolipelaamiseen. Nykyseikkailut, morpeista puhumattakaan, keskittyvät roolipelaamisen sijaan grindaamiseen. Garriottin tiimin näkemys palaa tarinankerronnan ytimeen: siihen, että roolipelissä ollaan joku muu. Hahmon roolissa eletään virtuaalisessa maailmassa. Koetaan ihmeitä ja tehdään mitä huvittaa. Leikitään kotia tai seikkaillaan, tutkitaan

Lord British, avaruuden lapsi

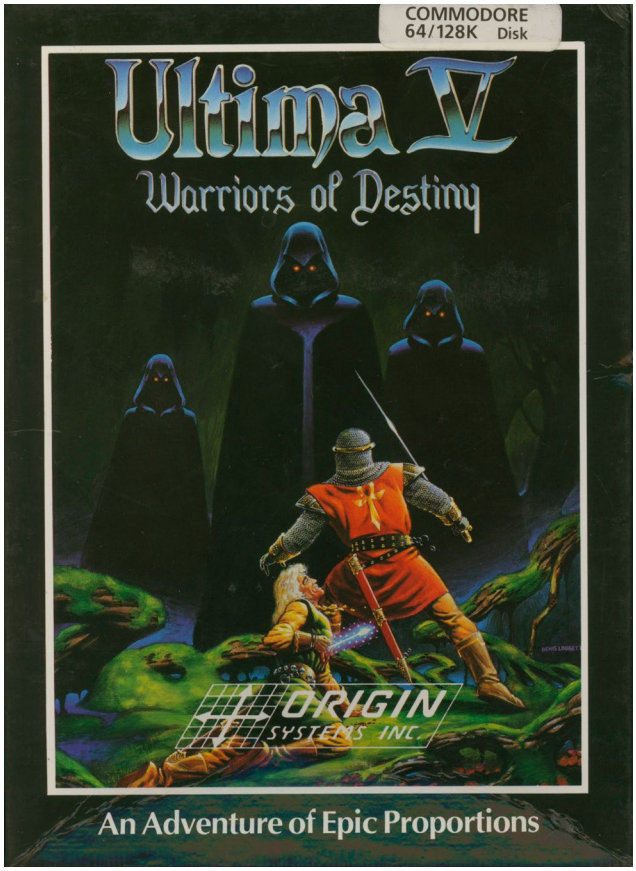
Ensimmäinen Ultima vei pelaajansa avaruusmatkalle. Myöhemmin Garriottin perustama Origin Systems teki monia avaruusseikkailuja ja NCSoftin leivissä mies loi scifimorppi Tabula Rasan. Vuonna 2008 Garriott suoritti avaruuslennon kansainväliselle avaruusasemalle ollen maailman kuudes avaruusturisti.

Garriottin elämää on fantasian ohella värittänyt rakkaus avaruuteen ja tieteeseen. Hänen isänsä, Owen K. Garriott, oli NASAn astronautti, joka työskenteli kahdella NASAn avaruusasemalla. Garriott onkin maailman toinen toisen sukupolven avaruuslentäjä.



Garriottin avaruuslennosta tehtiin myös dokumenttielokuva, jota on kutsuttu maailman kalleimmaksi kotivideoksi. 83-minuuttinen elokuva kuvaa niin avaruuslennon taustoja, koulutusvaiheen kuin avaruuslennon tapahtumiakin. Dokumentti kuvaa poikkeuksellisella avoimella tavalla venäläistä avaruusturistin koulutusvaihetta ja avaruuslennon arkipäivää. Erittäin suositeltavaa katsottavaa sekä Garriottin että yleensä avaruuslentojen faneille!

Elokuvan voi tilata muun muassa Amazonista. Lisätietoja:
http://www.imdb.com/title/tt1611990/



ja etsitään aarteita. Kohdataan lohikäärmeitäkin.

Seikkailussa ei ole lainkaan nykypelien tehtävärakenteita. Selkeiden tehtävien ja tavoitteiden sijaan pelaajia kannustetaan seuraamaan tarinan kaaria ja keskustelemaan maailmaa kansoittavien henkilöiden kanssa. Moni uusi tehtävä tai tarina voi löytyä niistä, liki huomaamatta. Shroud of the Avatar on peli, jossa ei ole kompasseja kohteisiin, ei kiinteitä hahmoluokkia, ei grindausta, ei rautalankaa.

Ei siis ihme, että Garriott lupaa pelimaailman olevan mahdollisuuksia täynnä. Samalla hän varoittaa, että pelaajien on kannettava vastuu teoistaan ja elettävä niiden seuraamusten kanssa. Peli, maailma ja sen asukkaat muistavat.

Shroud on vasta alkua. 30 vuotta sitten Garriott loi Ultimoiden maailman yksin. Nyt urakka alkaa alusta. Mikäli peli löytää pelaajansa, tulee Garriott tiimeineen työstämään uutta fantasiamaailmaa vielä pitkään, monen uuden pelin ja laajennuksen merkeissä. Ehkä Shroudista tulee uusi Ultima.

Viimeiseksi todelliseksi Ultima-seikkailuksi jäänyt Ultima IX sisälsi muodikasta pelinsisäistä mainontaa.

Peli sisältää yksinpelin, yhteistyö-henkisen moninpelin ystävien kanssa ja verkkomoninpelin. Early access -versio joukkorahoittajille julkaistaan vuodenvaihteessa, ja virallinen julkaisu ajoittuu vuoden 2014 loppuun.

Ho eyoh he hum

Mikä Ultimoissa viehättää? Yksi vastaus on pelisarjan merkitys koko tietokoneroolipelien genren synnylle. Aikana, jolloin seikkailupelit olivat lähes pelkästään tekstiseikkailuja, Akalabeth ja ensimmäinen Ultima tähtäsivät korkeammalle. Ne olivat aitoja roolipelejä.

Osa Ultimoiden lumosta tulee pelien maailmasta. Britannia kasvoi pelaajiensa kanssa peli peliltä, tasaisesti laajentuen. Se oli ensimmäinen todellinen virtuaalinen pelimaailma, rakkaudella rakennettu jatkumo, jossa lähes mikä tahansa tuntui olevan mahdollista. Ultimat olivat yhtä aikaa hiekkalaatikkomaisia avoimia seikkailuja ja vaihtoehtomaailmoja fantasian mahdollistamassa äärettömyydessä. Niiden käsikirjoittamattoman yllätyksellisyyden ja avoimuuden ansiosta jokainen koki tarinan eri tavoin.

Kuvailin kerran Ultimoita nykyaikaisiksi saduiksi. Tarinoiksi, joissa Hyvä ja Paha kamppailevat – ja Hyvä voittaa aina. Vaikka seikkailut suunnattiin varttuneille pelaajille, perusidea oli sama: oikeuden voitto ja heikkojen puolustaminen. Garriott koki velvollisuudekseen luoda peleihin syvempiä teemoja, joissa korostettiin suurempia hyveitä – kuten sitä, että vahvoilla on myös velvollisuuksia, ei ainoastaan oman käden oikeuksia. Ehkä siksi pelisarja on tuntunut aina niin oikealta, niin rehelliseltä.

Sekään ei haitannut, että Ultimat olivat pitkään todellisia tienraivaajia niin tarinankerronnaltaan kuin teknisesti. Todellisuuden rajoja työnnettiin luovalla 2D:n ja 3D:n yhdistelmällä, hiekkalaatikolla, pelaamisen vapaudella ja verkkomoninpelaamisen saralla. Siinä, missä Ultima Underworld (1992) loi ensimmäisen kolmiulotteisen roolipelikokemuksen, oli Ultima Online (1997) yksi ensimmäisistä todellisista verkkoroolipeleistä, jonka perintöä seuraavat tänään niin World of Warcraft kuin muutkin genren edustajat.

Kohokohtia ja kompasteluja

Ultima Onlinen alfa- ja beetatestivaihe kesti liki puolitoista vuotta. Pitkän viimeistelyvaiheen jälkeen pelaajat kerääntyivät juhlistamaan betan loppua pelimaailmaan. Originilla oli yllätys hihassaan: beta päättyi yllättäen maailmanloppuun. Ultimoiden arkkivihollinen, ulottuvuuksien välinen kauhistus, pahuuden perikuva ja maailmojen tuhoaja Guardian laskeutui Britanniaan. Maailma peittyi demoneihin ja kuolemaan, eikä edes Lord British säästynyt. Uskomaton tapa päättää beta ja avata maailma yleisölle!

Ultima Underworld: The Stygian Abyss (1992) oli yksi ensimmäisistä todellisista 3D-peleistä, ja äimistytti pelaajia jo ennen id Softwaren Doomia (1993) tai Quakea (1996). Samat tekijät loivat myöhemmin System Shockin (1994) ja Thiefin (1998) kaltaiset mestariteokset.

Ultima VIII Part Two: Serpent Islen (1993) ei alkujaan edes pitänyt olla Ultima. Sen kehitys kuitenkin junnasi ja pelimoottoria päätettiin käyttää Ultima-seikkailuun. Aikataulut pettivät pahemman kerran ja rahoittajat hermostuivat, minkä johdosta peli tuupattiin kauppoihin hätäisesti kokoonkursittuna, sisällötään riisuttuna versiona. Kiireen takia teokseen jäi runsaasti bugeja ja tarinallisia aukkoja, eikä kaikkia sivutehtäviä voinut edes ratkaista.

Ultima Onlineen (1997–) tehtiin interaktiivinen maailmasimulaatio, jossa kaikella oli tarkoituksensa ja kaikki vaikutti kaikkeen, aina luonnon kiertokulusta lähtien. Työläs virtuaalimaailman mallinnus ei kuitenkaan ehtinyt edes käynnistyä, kun pelaajat säntäsivät metsiin ja tappoivat kaiken. Aivan kaiken. Eräänkin kerran näin yhden ainokaisen pupun perässä puoli tusinaa pelaajaa. Lopulta maailmasimulaatio poistettiin, koska pelimaailmaan levittäytyneet ihmiset tuhosivat ekosimulaation lähtökuoppiinsa. Kukaan ei edes huomannut sen olemassaoloa.

Mitä tapahtuikaan, kun Avatar nautti taikasieniä Ultima IX:ssä?

Faneille Ultima-sarja on kuollut, mutta legendan nimi elää kahdessa uudessa mikromaksupelissä. **Lords of Ultima (2010)** on käpyinen ilmaispelattava MMO-strategia, ja iOS-alustalle saapunut **Ultima Forever** -mobiilimorppi **(2013)** on eräiden pelaajien mukaan sarjan historian masentavin, idiooteille tehty rimanalitus.

Säteilyaavikoiden kutsu

Wasteland 2 nousi radioaktiivisesta haudastaan.

Menestynyt joukkorahoituskampanja herätti henkiin myös toisen klassikon. Science fiction -roolipeli Wasteland (1988) oli aikanaan suuri menestys, joka vaikutti merkittävästi roolipelien kehittymiseen ja tarinankerrontaan. Se antoi nostetta kovan luokan science fictionin suosiolle tietokoneroolipelien piirissä, mutta ansioistaan huolimatta pelin jatko-osan syntyä on odoteltu epäonnisten tähtien alla.

Wasteland oli amerikkalaisen pitkän linjan pelisuunnittelija Brian Fargon (1962) käsialaa. Fargo teki ensimmäisen kaupallisen seikkailupelinsä, Demon's Forgen, jo 1981 ja perusti Interplay-pelitalon 1983. Moni Fargon teoksista muistetaan seikkailu- ja roolipeliklassikkoina, joista mainittakoon Bard's Tale -sarja (1985), Borrowed Time (1986), Dragon Wars (1989), Neuromancer (1989) ja Star Trek: 25th Anniversary (1992).

Fargon peleistä legendaarisin on silti Wasteland. Ydintuhon jälkeiseen maailmaan sijoittuva roolipeli rakensi ainutlaatuisen hienosti käsikirjoitetun ja kyseenalaistavan tarinan romahduksen jälkeisestä maailmasta asukkaineen. Se oli kuin sarkastinen, älykäs videopeliversio Mad Max -elokuvista, jonka taitavaa tarinankerrontaa ja maailmankuvaa on sittemmin lainattu liki kaikissa ydintuhokuvauksissa, ei vähiten Fallout-sarjassa. Jos Wastelandin oikeaa jatko-osaa saatiinkin odottaa 25 vuotta, Falloutit ainakin tekivät parhaansa muiston ylläpitämisessä.

Veteraanikehittäjä Brian Fargo teki ensimmäisen kaupallisen pelinsä vuonna 1981.

Paluu ydinaavikoille

Wastelandin ansioita olivat muun muassa ydinaavikon outoja tapahtumia tutkivien pelihahmojen kuvaaminen inhimillisinä hahmoina, jotka eivät aina käsittäneet pelaajan kotkotuksia tai suostuneet noudattamaan kummallisia käskyjä. Pelimaailma oli dynaaminen ja muuttui jokaisella pelikerralla.

Wasteland 2 noudattaa esikuvansa mallia tyylikkäästi, nykypäivän vaatimuksia noudattaen. Se sijoittuu aikaan 15 vuotta ensimmäisen pelin ja noin sata vuotta maailman raunioittaneen atomisodan jälkeen.

Pelillisesti Wasteland 2 tuntuu veteraaneille tutulta, mutta nykypelaajatkin huomioidaan. Hahmojärjestelmä on pohjaltaan sama kuin 25 vuotta sitten. Sen taustalla on tieteiskirjailijanakin tunnetun Michael A. Stackpolen Mercenaries, Spies and Private Eyes -lautapelin hahmojärjestelmä. Systeemi on erittäin toimivan oloinen ja mahdollisesti yksi hardcore-tasoisimmista, mitä uusista peleistä löytyy. Syystäkin, sillä pelaajien rahoittamaa peliä ei tarvitse siloitella julkaisijoiden pelkojen mukaan.

Teoksen suurimmat avut löytyvät tarinankirjoituksesta ja kauaskantoisuudesta. Se on kirjoitettu esikuvansa tapaan täyteen moraalisia valintoja ja päätöksiä, joilla on sekä välittömiä että kauaskantoisia seuraamuksia. Nämä vaikuttavat sekä seikkailuun että maailman ja asukkaiden suhtautumiseen pelaajan ryhmään – ja tietenkin pelaajaan itseensä. Mihin henkinen kanttisi venyy? Kestätkö päätöstesi aiheuttamat, kenties yllättävät ja kauhistuttavat seuraukset? Kääntyivätkö hyvät tarkoitusperät nurinniskoin?

Wasteland 2 vaikuttaa peliltä, joka on vahvasti vanhan koulun edustajien mieleen. Sen kompromissittomuus näkyy muun muassa tekstin paljoudessa. Kun rahoja ei törsätä dialogin puhumiseen, voidaan tekstiä kirjoittaa arastelematta. Ongelmiin löytyy paljon erilaisia ratkaisuja, ja pelijärjestelmän avoimuuden ansiosta pelaajat keksivät varmasti odottamattomia temppuja.

Seikkailun koosta puhumattakaan! "Tämä on ISO peli", Fargo rykäisee allekirjoittaneelle ja jatkaa, että se on "paljon isompi kuin Fallout 1 tai 2".

"Useimmat pelaajat eivät koskaan ehdi nähdä kaikkea", Fargo lupaa ja kehuu, että eriävien tarinapolkujen ja vaihtelevien valintojen ansiosta tekemistä riittää varmasti useaksi läpipeluukerraksi.

Wasteland 2 ilmestyy vuoden 2014 aikana.

Wastelandin kirous

Vaikka Wasteland oli rakastettu hitti, ei peliala hellinyt sen jatko-osia. Ensin se jäi Electronic Artsin politiikan jalkoihin. Yhtiön tapasi teettää pelit ulkopuolisella studiolla ja jos se menestyi, jatko-osa tehtiin EA:n sisällä. Niin nytkin. 1990 ilmestyi Fountain of Dreams, mutta sitä ei lopulta markkinoitu Wastelandin jatko-osana. Osasyynä tähän oli se, ettei Fountainin tekijöissä ollut ketään Wastelandin tiimistä, eikä peli yltänyt Fargon luomuksen tasolle.

Lisenssin menetettyään Fargo ja Interplay tekivät Fountainin rinnalla omaa jatko-osaansa nimeltä Meantime. Se jäi kesken, kun kotimikromarkkinat siirtyivät ryminällä 16-bittiseen aikakauteen, eikä peliä kannattanut siirtää uudelle alustalle.

Fargo palasi ydinaavikoille vuonna 1997 toisen klassisen roolipelin, Falloutin, parissa. Sarja ei kuitenkaan lainauksistaan huolimatta ole jatkoa Wastelandille.

Fargo on kertonut kuvitelleensa 20 vuoden ajan jokaisen pelinsä jälkeen, että nyt on oikea hetki tehdä Wasteland 2. Aina turhaan. Paluu ydinaavikoille ei löytänyt rahoittajaa edes Fallout 3:n tai New Vegasin menestyksen jälkeen, vaan vasta suoraan faneihin vetoaminen mahdollisti pelinteon.

Ei pelattavissa

Tapio Berschewsky

Kulttuuria pitää säilöä, mutta kenelle sitä säilötään? Uusi pelaajasukupolvi on lukutaidoton sille, mikä ennen oli selvää.

Kymmenen vuotta takaperin Schipholin lentokentällä törmäsin aarteeseen. Pienestä putiikista irtosi halvalla Game Boy Advancen versio Manic Minerista. Matthew Smithin tasohyppely oli ennestään tuttu Commodore 64 -julkaisusta.

Manic Miner oli niin vaikea, etten 80-luvulla päässyt ikinä edes kuudenteen kenttään. Jo ensimmäinen ruutu vaati useamman yrityksen, jotta kömpelösti liikkuvan hahmon sai ohjattua loppuun asti.

Pääsin Game Boylla pidemmälle, mutta en kovin montaa kenttää. Tämä sai ajattelemaan, kuinka helpoilta nykypelit jo silloin tuntuivat. Manic Minerin design on armoton. Genren nykypelien kanssa aloittaneelle se olisi suorastaan mahdoton.

Kasarilla suunniteltiin myös tasohyppelyjä, joiden haaste ei perustunut yliampuvan vihamieliseen ympäristöön ja kiikkerään hahmoon. Aikanaan Super Mario Bros 1 ei tuntunut vaikealta. Ensimmäinen kenttä meni läpi heti ja itse pelikin muutamassa illassa.

> *Oppilaat eivät selviä Ultima IV:stä.*

Helposti ohjattavan hahmon ja mainion kenttäsuunnittelun olettaisi toimivan aivan yhtä hyvin nyt. Näin ei kuitenkaan ole.

Kesäkuussa 2013 Nintendon toimitusjohtaja Satoru Iwata kertoi osakkeenomistajille tyrmistyttävän tarinan. Nintendon pelit ovat nykyään helpompia kuin 80-luvulla, koska uudet pelaajat eivät selviä SMB 1:stä.

Nintendon testissä vain 10 % uusista yrittäjistä pääsi ensimmäisen kentän läpi. Noin 70 % kuoli ensimmäiseen viholliseen. Monet pelkäsivät kolikoita ja koettivat välttää niitä.

Ihmiskunnan ylpeys

Tässä lehdessä muistelemme Ultima-roolipelisarjaa. Itselleni sen ikimuistoisin osa on neljäs. Se oli monipuolisempi kuin mikään näkemäni peli. Pelaaminen vaati manuaalien lukemista. Kymmenvuotiaalle pelin englanti oli vaikeaa, joten sanakirja pysyi mukana.

Alkuun pääsi heti, mutta läpäisy odotutti hetken itseään. Tarkoitus ei ollut voittaa pahaa pääörkkiä vaan kilvoitella täydelliseksi ihmiseksi, Avatariksi. Peli mittasi pelaajan toimien hyvyyttä, ja vain kirkasotsainen näki lopputekstit.

Muistan, kun ensimmäistä kertaa sain haltuuni piraattilaivan ja matkustin sillä toiselle saarelle, jonka kauppa myi maagisia välineitä. Koulukaverit väittivät minun valehtelevan. Heistä oli käsittämätöntä, että peli voisi olla niin laaja ja hieno.

Englannintaitoiselle nykyteinille Ultima IV:ssä ei kai pitäisi olla mitään mahdotonta. Iso avoin maailma aarteita pursuavine luolastoineen, loitsuineen, pulloineen ja taisteluineen on tuttua kamaa. Näinhän se Skyrimkin toimii.

Wabashin korkeakoulun professori ja Brainy Gamer -blogin perustaja Michael Abbott kirjoitti vuonna 2010, että hänen oppilaansa eivät selviä Ultima IV:stä.

Abbott tiputtaa kurssilaisensa viikoksi Britanniaan pdf-manuaalien kanssa, vailla muuta opastusta. Kolme vuotta sitten uudet oppilaat eivät enää ymmärtäneet peliä. He eivät tienneet, miten edetä tai mikä on tavoite.

Manuaaleista kysyttäessä luokka ihmetteli, eivätkö ne olleetkaan vain pakollista laatikon täytettä. Kun Abbott oli ilmaissut, että pelaajien oletettiin lukevan ne ennen pelaamista, olivat oppilaat tyrmistyneitä.

Klassinen rope oli muuttunut pelikelvottomaksi. Siinä ei ollut oppaita, ei tutkaa eikä tehtävälistaa. Edistymisen tunne, joka syntyy omin päin maailmaa tutkimalla, jäi saavuttamatta, koska oppilaat olivat tottuneet kädestä pitelyyn.

Onko tämä paha?

Mainitut pelit eivät olleet pienen hardcore-yleisön indiepelejä vaan isoja julkaisuja. Niitä pelasivat kaikki, jotka genreistä pitivät. Nyt ne ovat pelikelvottomia valtaosalle uusista yrittäjistä.

Pelien kieli on muuttunut kuluneen 30 vuoden aikana melkoista vauhtia. Näistä esimerkeistä päättelemällä voisi sanoa, että se muuttui tunnistamattomaksi vain yhden sukupolven aikuistumisen aikana. Ainakin uusille pelaajille, taaksepäin katsoen.

Kohtaamme siis ehkä ensi vuosikymmeneen mennessä pelaajasukupolven, jolle Monkey Island, Wing Commander ja Civilization ovat epäintuitiivisia ja mystisiä kokemuksia, joista he eivät saa selvää. Kunhan siskoni lapset alkavat olla peli-iässä, taidan suorittaa muutaman ihmiskokeen.

Seuraavassa Skrollissa alkaa juttusarja modernin pelisuunnittelun keinoista. Ovatko keinot universaaleja vai aikaan sidottuja? Tottumusten summa määrittää, mikä on yksinkertaista ja mikä ei. Onko tulevaisuus Cookie Clickerin?

Fanikäännösprojektit

Kun vain itse tehty on kyllin hyvää.

Pelifanit ovat aina löytäneet keinoja pelaamiseen ja pelikokemusten jakamiseen. Joskus tiellä on kielellisiä ja kulttuurisia muureja, mutta tosifaneille ne ovat vain mukava haaste!

Teksti: Visa-Valtteri Pimiä Kuva: Tuukka Virtaperko

Kulttuuriset erot länsimaiden ja Aasian maiden pelihistoriassa ovat aiheuttaneet sen, että monia aasialaisia pelejä ei ole lainkaan julkaistu länsimaissa, lähinnä kaupallisista syistä. Käännösten tekeminen kun on kallista, etenkin jos pelin myynti jää vähäiseksi. Niinpä fanit ovat toimineet oma-aloitteisesti. Jo 90-luvulta saakka he ovat hakkeroineet alkuperäisiä pelejä ja kääntäneet niitä toiselle kielelle. Heidän tavoitteenaan on käännöspelien saatavuuden parantaminen ja pelikokemusten jakaminen muillekin.

Fanikäännösprojektien historia ja ryhmien organisointi

Tietokone- ja konsolipelien laajamittaiset fanikäännökset ovat ainutlaatuisesti internetin synnyttämä ilmiö. Pelejä käännettiin pienissä piireissä jo 80-luvun lopussa ja 90-luvun alkupuolella, mutta siitä ei jäänyt juurikaan todisteita. Parhaiten tunnetut esimerkit olivat piraattijulkaisuja ja syntyivät Nintendon Famicom/NESin ja muiden pelilaitteiden harrastus- ja piraattipiireissä. Julkaisut sisälsivät vain vähän tekstiä.

Vasta internetin ja emulaattoreiden yleistymisen myötä tulivat pinnalle julkisesti dokumentoidut ja maailmanlaajuiset käännösprojektit, joissa huomio keskittyi erityisesti tarinasisältöön ja käännöksen laatuun. Toiminta käynnistyi, kun laiteharrastajat löysivät ensimmäisistä suurista ROM-tiedostokokoelmista pelihelmiä, joita ei ennestään länsimaissa tunnettu. Teknisesti kyvykkäät ja japanin kieltä osaavat harrastajat synnyttivät lukuisia projekteja pelien kääntämiseksi englannin kielelle. Tässä artikkelissa keskitytään näihin myöhempiin, kunnianhimoisempiin ja teknisesti haastavampiin projekteihin.

Jo alkuvaiheessa käännösryhmät koostuivat useista erilaisista jäsenistä. Ohjelmoijien vastuulla on pelin ohjelmakoodin muuttaminen ja alkuperäisten, vierasta kieltä sisältävien grafiikoiden ja tekstien korvaaminen käännetyillä. Usein ohjelmoijat kehittävät myös työkaluja niille, joilla on vähemmän teknistä osaamista tai joilla on asiantuntemusta muilla aloilla.

Kääntäjien työhön kuuluu alkuperäisen pelin teksti- ja grafiikkasisällön kääntäminen. Tämä vaatii yleensä kattavaa ymmärrystä lähdeteoksen kulttuurikontekstista. Kulttuurinen tietous on tärkeää, jotta länsimaiselle kohdeyleisölle vaikeasti avautuvat viittaukset voidaan korvata tutummilla vastineilla.

Graafikoiden tehtävään kuuluu alkuperäisteoksen vieraskielisten kuvaelementtien muokkaaminen ja uusien kirjaintyyppien luominen. Grafiikassa erityinen haaste on alkuperäisen pelin visuaalisen kielen ja tyylin säilyttäminen. Käännettyihin kuviin täytyy sisällyttää pelin ja tarinan ymmärtämisen kannalta oleellinen tieto.

Lisäksi projekteissa voi olla mukana tekstikäsikirjoitusta parantelevia ja muokkaavia editoijia sekä testaajia, joiden tehtävänä on testata käännöspelien toimivuus ja ymmärrettävyys.

Fanikääntämisen taide ja tiede

Muutamia poikkeuksia lukuun ottamatta fanikäännöksiä tehdään japanilaisista peleistä, joista julkaisija ei ole tehnyt virallista käännöstä. Japanissa julkaistaan pelejä paljon, mutta länsimaiset maahantuojat vain valitsevat niistä omille asiakkailleen sopivia ja entuudestaan tuttujen pelien jatko-osia. Kulttuurierot toki vaikeuttavat kääntämistä ja saattavat tehdä sen taloudellisesti kannattamattomaksi.

Faneja tällaiset esteet eivät kuitenkaan haittaa. Koska pelien lähdekoodeja ja -tiedostoja ei yleensä ole saatavilla, tarjoaa pelien kääntäminen myös ainutlaatuisia teknisiä ja tarinankerronnallisia haasteita. Niitä kääntäjäryhmä ratkoo yhteisvoimin.

Moonspeak – monien merkistöjen magiaa

Japanilaiset pelit käyttävät sekä äänneettä ideogrammimerkistöjä. Jälkimmäisen etuna on se, että peliruudussa pienellä kuvapistemäärällä voi ilmaista suuren määrän informaatiota. Kahdeksanbittisillä alustoilla ideogrammimerkistön käyttäminen ei yleensä ole mahdollista, koska muistia on vähän. Siksi pelien käyttöliittymässä ja dialogissa suosittiin yleensä äännemerkistöjä, ja sellaisen korvaaminen länsimaisilla kirjaimilla onkin yleensä huomattavasti pienempi työ kuin vastaavien ideogrammien.

Pelkän valikkotekstin ja dialogin kääntäminen ei kuitenkaan aina riitä. Peleissä saattaa esiintyä japanilaista tekstiä myös grafiikoissa, kuten pelin nimiruudussa, taustagrafiikassa ja onomatopoeettisissa äänitehosteissa, joiden idea on lainattu mangasta. Tämän takia kääntäminen vaatii usein myös taitavaa graafikkoa, joka osaa jäljitellä alkuperäisten taiteilijoiden tyyliä. Monille käännösprojekteille saumaton graafinen kokemus on kunnia-asia ja merkittävä tekijä käännöksen laadussa.

Taistelu muistin vähyyttä vastaan

Englanninkieliset ilmaukset ovat usein monisanaisempia kuin alkuperäiset japaninkieliset, mistä seuraa suurempi muistinkulutus. Varsinkin konsolipeleissä rajoitteena on pelimoduulin ROM-muistin määrä ja pelin muistinkäyttöä voi joutua miettimään uudelleen. Joskus se vaatii pelikoodin kekseliästä muokkaamista tai pelin alkuperäisen fyysisen muistikonfiguraation korvaamista suuremmalla vaihtoehdolla.

Onpa tehty sellaisiakin käännöksiä, jotka eivät edes toimi alkuperäisellä konsolilla vaan pelkästään emulaattorin sallimissa ”valheellisissa” rajoissa. Näissä ei huomioida laitteiston ominaisuuksia joko tarkoituksella tai naiivin toteutuksen vuoksi.

Monet varhaiset pelikäännökset olivat laadultaan heikkoja, koska käännösprojekteja aloittelevat amatööriohjelmoijat eivät useinkaan osanneet lisätä peliin länsimaisten kielten vaatimaa tilaa. Niinpä he usein typistivät käännöksestä tekstiä. Joskus pelistä jätettiin pois tarinalle tärkeitä osia.

Moni alkujaan heikkolaatuinen käännös julkaistiin myöhemmin uudelleen, kun toiset ryhmät olivat parannelleet aiempia käännöksiä. Kun yhteisöjen tekninen osaaminen, työkalujen laatu ja ryhmien keskinäinen tiedonvaihto paranivat vuosien saatossa, parani myös käännösten laatu huomattavasti. Näin kompromissien määrä väheni merkittävästi. Nykyisin fanikäännökset ovat yhtä hyviä tai jopa parempia kuin isolla rahalla tuotetut kaupalliset käännökset.

Onnistuu se muillakin kielillä

Pelejä ei käännetä aina pelkästään englanniksi, vaan niitä on tehty jopa arabiaksi. Joel ”Bisqwit” Yliluoma on kääntänyt muun muassa Tales of Phantasian, Chrono Triggerin ja Castlevania II – Simon’s Questin suomeksi. Näitä teoksia voi tarkastella ja ladata Yliluoman sivuilta.

Kulttuurisidonnaisuus

Kuten käännöstyössä yleensäkin, on japanilaiselle kulttuurille ominaisten käsitteiden kääntäminen omanlaisensa haaste. Internet on kuitenkin mahdollistanut mannertenväliset yhteistyöprojektit, joissa japanilaiset jäsenet auttavat muita ymmärtämään japanilaisia käsitteitä. Toisaalta animen ja japanilaisen elokuvan maailmanlaajuinen suosio on tehnyt monista aiemmin vieraista käsitteistä arkipäiväisiä myös länsimaiselle yleisölle.

Käännökset toiseen suuntaan eli englannista japaniksi ovat äärimmäisen harvinaisia. Yhtenä syynä saattaa olla se, että japaninkielisiä pelejä julkaistaan virallisestikin suuret määrät ja ne kattavat merkittävän osan runsastekstisistä länsimaisista peleistä. Japanilaisilla on myös pitkä suhde videopeleissä käytettyyn englantiin, minkä ansiosta pelisanasto on monille tuttu, vaikka englantia ei muutoin hyvin osaisikaan.

Linkkejä:

- Joel Yliluoman suomenkielisiä fanikäännöksiä http://bisqwit.iki.fi/gametrans.html
- Phantasian Productions: http://www.tales-cless.org/
- Policenauts Translation Project: http://policenauts.net/english/
- Touhou Patch Center: http://thpatch.net/wiki/Touhou_Patch_Center
- ROMHacking.net (käännöstietokanta): http://romhacking.net

Tunnettuja käännösprojekteja

Policenauts Translation Project (1996)

Monien fanikääntäjien unelmaprojektina oli pitkään CD-helmi Policenauts. Pelin on luonut Hideo Kojima, joka tunnetaan Metal Gear -sarjastaan. Peli jäi melko tuntemattomaksi Japanin ulkopuolella. Se oli scifiseikkailu, joka sisälsi valtavasti tekstiä sekä puhuttua dialogia alusta loppuun saakka. Se kuvitettiin animetyylisesti videoilla. Kojiman peleistä se oli ainoa, jota ei koskaan lokalisoitu länsimaihin.

Sega Saturn -version virallinen englanninkielinen versio luvattiin kauppoihin jo 1996, mutta sitä ei kuitenkaan julkaistu koskaan. Niinpä fanit aloittivat oman käännösprojektinsa, jota tehtiin yli vuosikymmen. Varsinainen käännös valmistui vasta vuonna 2007, mutta käsikirjoitus ja käännetyt grafiikat piti vielä upottaa pelimoottoriin. Tähän tarvittiin taitavaa ohjelmoijaa, joka tekisi pelin tiedostoihin ja ohjelmakoodiin tarvittavat muutokset. Niinpä projekti pysähtyi noin vuodeksi.

Something Awful -sivuston käyttäjä Michael ”slowbeef” Sawyer aloitti kuitenkin PlayStation-version kokeilun syksyllä 2008. Slowbeef onnistui purkamaan pelikoodin, ja pelin fanikäännöksen ensimmäinen versio julkaistiin sopivasti Kojiman syntymäpäivänä vuonna 2009. 13 vuoden ajan viilattu Policenautsin käännös onkin yksi korkealaatuisimmista fanikäännöksistä, jolle monet virallisetkin käännökset kalpenevat.

Touhou Project (1996)

Varmaankin kaikkien aikojen suosituin Doujinsoft-sarja, länsimaalaisittain karkeasti tulkittuna indiepelisarja Touhou Project, sisältää pääkaanonissaan jo kymmeniä pelejä. Niihin lasketaan kaikki alkuperäisen kehittäjän, Zunin Shanghai Alice -studion teokset. Lisäksi fanit ovat tehneet pelejä, joissa käytetään samoja hahmoja ja joskus myös samoja peliominaisuuksia.

Touhou-sarjan ensimmäinen peli oli eräänlainen Breakout-klooni, mutta myöhemmissä luomuksissa on keskitytty danmaku-pelityyliin, joka käännetään englanniksi bullet helliksi eli luotihelvetiksi. Luotihelvetti-termi onkin äärimmäisen osuva nimitys ammunta- ja väistelypelille, jossa monimutkaiset ammuskuviot tavoittavat omanlaisensa esteettisen kauneuden. Peleissä selviämisen elinehtoja ovat sekä ulkoa opettelu että toistuvien kuvioiden tarkka ennalta lukeminen.

Kaikille sarjan peleille on yhteistä taikamaassa seikkailevat taikatytöt, keijut, noidat ja ties mitkä muut yliluonnolliset olennot, joiden tärkein hyökkäysmetodi on epileptisten kuviomerten taikominen pelaajahahmoa vastaan. Kääntäjille Touhou-peleissä on ollut työtä lähinnä tarinaosuuksissa ja käyttöliittymäteksteissä. Niinpä Touhou-pelejä on pelattu jo ennen fanikääntäjien julkaisuja, sillä kielimuuri ei juuri pelaamista hidasta.

Tales of Phantasia (1995)

Tales of Phantasia on Wolf Team -ryhmän kehittämä japanilaistyylinen, fantasiamaailmaan sijoittuva JRPG-roolipeli Nintendon SNES-konsolille. Emulaattoreiden yleistyessä tämä teknisesti ja sisällöllisesti näyttävä teos sai paljon huomiota, mutta sen kääntäminen oli erityinen haaste, koska peli oli massiivinen ja sisälsi japaninkielistä puhetta.

Lopullinen käännös ei koskenut puheeseen mutta käänsi kaiken tekstin ja grafiikan. Pelistä julkaistiin myös dubattu versio, jonka ääninäyttelijöiden taidot ovat jokseenkin kyseenalaiset. Vuosia myöhemmin SNES-käännöksestä julkaistiin paranneltu PlayStation 1 -versio, jota tehtiin useita vuosia. Tämä versio on vieläkin kunnianhimoisempi, sillä se sisältää puheiden tekstityksen ja PS-version lisäsisällön käännökset. Ensimmäinen virallinen käännös pelistä julkaistiin vasta vuonna 2006 Gameboy Advancelle.

JUHA JA TOMMI ETSIVÄT TYÖKAVERIA TOSITARKOITUKSELLA

Tulin Vincitille 2009, jolloin meitä oli noin 20. Siitä yritys on mennyt huimasti eteenpäin ja nyt meitä on jo 80! Nykyisin vedän Vincitin mobiilitiimiä. Ollaan toteutettu esimerkiksi metsästystutka, lyhytelokuvapalvelu ja muumipeli. Meinataan hukkua mielenkiintoisiin projekteihin joten apuvoimia kaivataan kuumeisesti.

– Juha R.

Olen ollut puolisen vuotta Vincitillä kehittämässä web-palveluita. Projektit ja niissä käytetyt teknologiat ovat olleet tosi monipuolisia. Työn ja opiskelun yhteensovittaminen onnistuu helposti, sillä TTY on kävelymatkan päässä ja Vincitillä ollaan joustavia työaikojen suhteen.

– Tommi L.

careers@vincit.fi
www.vincit.fi

vincit
Innokas ohjelmistotalo.

Vuoden yritys 2013
– OP-Pohjola & Kauppalehti

Tekstipelien lumo
– Pulmapeleistä interaktiiviseen fiktioon

Fotorealismiin yltävän grafiikan aikakaudella ajatus pelkkää tekstiä sisältävästä pelistä voi tuntua oudolta kunnes muistaa, että maailman tehokkain näytönohjain löytyy korvien välistä.

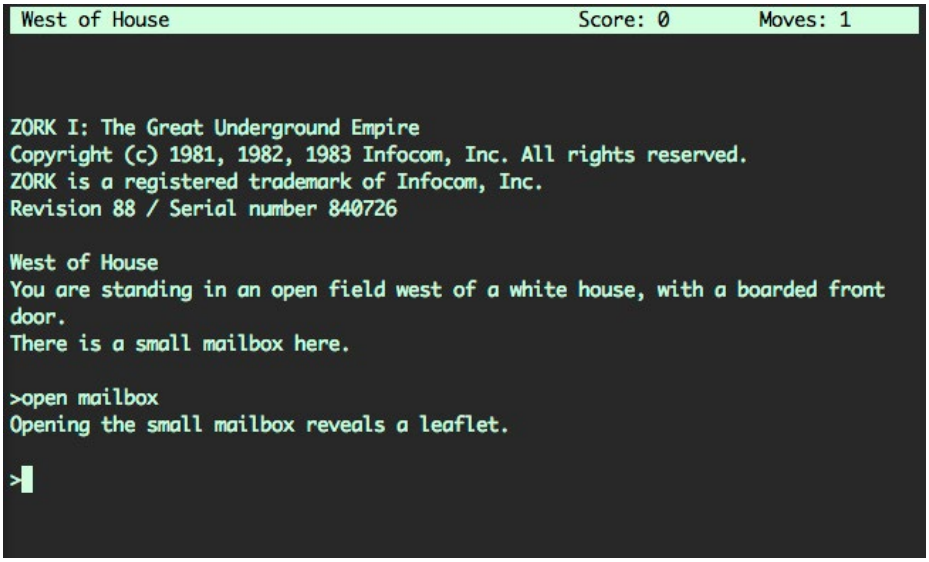
Teksti: Janos Honkonen Kuvat: Mikko Rasa, Jukka O. Kauppinen, Janos Honkonen

80-luvulla tietokonegrafiikka oli vielä varsin alkeellista, eikä kotimikroilla pystynyt luomaan kovin näyttäviä kohtauksia tukeutumatta pelaajan mielikuvitukseen. Ei ihme, että kyseinen vuosikymmen oli tekstipelien kulta-aikaa. Muutamalla lauseella pystyi luomaan mielikuvia massiivisista avaruustaisteluista, joiden toteuttaminen tietokonegrafiikalla oli vain muutaman kyberpunkkarin haaveunta. Tekstipelejä ei nykyisin pelikauppojen hyllyiltä löydy, mutta lajityyppi on kaikkea muuta kuin kuollut, kiitos innokkaan ja aktiivisen harrastajakunnan.

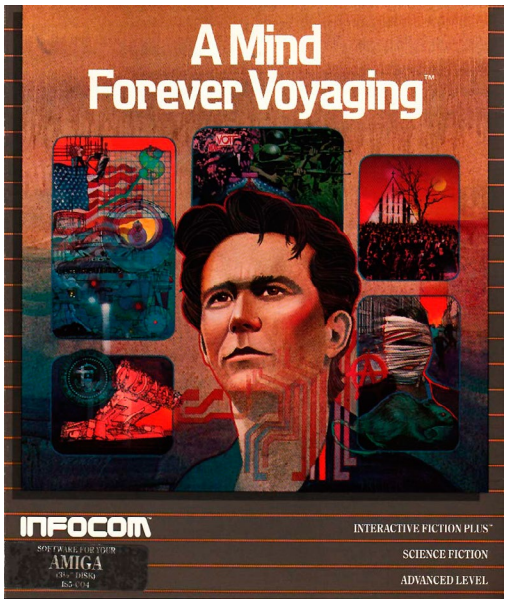
Tekstipeli, anteeksi mikä?

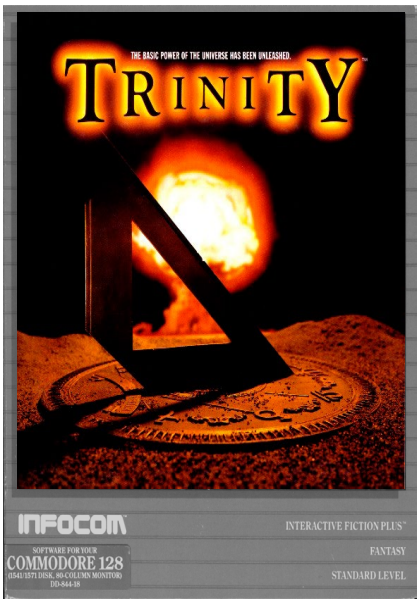
Tekstipeleissä pelaaja saa tekstimuotoisen kuvauksen pelihahmon ympäristöstä, tilanteesta ja maailmassa tapahtuvista asioista, ja pelaaja ohjaa hahmoaan tekstikomennoin. Legendaarinen Zork, yksi genren varhaisimmista edustajista, alkaa tilanteesta, jossa peli kertoo hahmon olevan avoimella niityllä valkoisen talon länsipuolella. Talon ikkuna on suljettu laudoilla ja sen edessä on pieni postilaatikko. Pelaaja voi reagoida tilanteeseen kirjoittamalla yksinkertaisen komennon, vaikkapa ”avaa postilaatikko” tai ”katso taloa”, minkä jälkeen peli kertoo, mitä toiminnasta seuraa.

Yksinkertaisimmillaan pelimaailman kuvaukset saattavat olla parin lauseen mittaisia. Mutta vaikka kuvaukset saattoivatkin olla pitkiä, oli parseri varsin alkeellinen. Parseri on ohjelmakomponentti, joka kääntää pelaajan luonnollisella kielellä kirjoittamat komennot tietokoneen ymmärtämään muotoon. Etenkin kotimikroaikaan parserit ymmärsivät usein vain kahden sanan komentoja, kuten ”open door” tai ”kill spider”. Kehit-



Nykypeleihin tottuneille tekstipelit näyttävät karuilta.





Tekstipelit eivät olleet alkumetreilläkään pelkkää kevyttä viihdettä, vaan niissä käsiteltiin ydintuhoa, politiikkaa ja muita raskaampia teemoja.

tyneemmät parserit selvisivät jo varhain vaikeammista komennoista, kuten ”plant the pot plant into the plant pot and drop it”.

Pelimaailma on jaettu ”huoneisiin”, jotka eivät ole varsinaisia huoneita maailman sisällä vaan ne voivat olla yhtä hyvin avaruusaluksen komentosilta, vuoren huippu kuin ruohotasanko. Liikkuminen tapahtuu perinteisesti ilmansuuntien mukaan sekä ylös ja alas. Vaikka tämä ei olisi tarinan tasolla luontevin mahdollinen tapa, sitä käytetään usein edelleen siksi, että käytäntö on pelityypin harrastajille tuttu.

70-luvun luolaseikkailu loi uuden pelityypin

Maailman ensimmäinen tekstipeli luotiin vuonna 1975. Sen nimi oli ytimekkäästi Adventure, joskin myöhemmin se on tunnettu myös nimillä Colossal Cave ja Colossal Cave Adventure. Luolaharrastaja Will Crowtherin luoma peli oli virtuaaliversio Yhdysvalloissa Mammoth Caven kansallispuistossa sijaitsevasta Colossal Cave -luolastosta.

Aitoa luolaa höystettiin fantasiahahmoilla, kuten kääpiöillä, varastelevalla merirosvolla ja peikolla. Crowther loi pelin avioerossa vieraantuneille lapsilleen ja pyrki yhdistämään siinä omia kiinnostuksen kohteitaan, kuten luolailun ja Dungeons & Dragons -roolipelin. Luonnollista kieltä käyttävien komentojen tarkoitus oli tehdä pelistä helposti lähestyttävä myös tietokonetaidottomille.

Adventure sisälsi kaikki nykypeleistä tutut elementit: tekstimuotoisen kuvailun ja parserin, yksinkertaiset puzzlet sekä nykyisissä kirjallisissa peleissä harvemmin käytettävän pisteytyksen. Se loi koko lajityypille pohjan, jota käytetään yhä tänään.

Luolastoista Linnunradan käsikirjaan ja pehmopornoon

80-luvun alkupää oli kaupallisten tekstipelien kulta-aikaa. Silloin syntyivät sellaiset klassikot kuin Zork-sarja, The Hitchhiker’s Guide to the Galaxy, Trinity ja Mind Forever Voyaging. Näiden kaikkien takaa löytyy alan merkittävin pelijulkaisija Infocom, joka vuonna 1979 kehittämä Z-machine-järjestelmä on käytössä edelleen.

Britannian markkinoilla merkittävimmät julkaisijat olivat Level 9 Computing ja Magnetic Scrolls. Osa tekstipeleistä jatkoi Adventuren ja Dungeons & Dragonsin viitoittamalla kevyellä seikkailulinjalla, mutta pelityypin kirjalliset ulottuvuudet ymmärrettiin jo varhain. Steve Meretzkyn kirjoittama A Mind Forever Voyaging on vakavamielinen ja poliittinen tarina, jossa pelaaja kontrolloi maailman ensimmäistä tekoälyä nimeltään PRISM. Ydintuhon pelkoa ja poliisivaltiota käsittelevä peli oli suoraa kritiikkiä Ronald Reaganin ajan politiikalle. Samoja teemoja käsitteli Brian Moriartyn Trinity, joka sekoittaa teemoja proosarunoudesta, ydintuhosta ja sodan turhuudesta fantasiaelementteihin.

Kevyemmän puolen tekstipeleissä luotiin humoristinen ja fantastinen kerrontatapa, joka elää edelleen graafisten seikkailupelien puolella. Esimerkiksi Monkey Island -pelisarja jatkaa suoraan huumoritekstiseikkailujen perintöä.

80-luvulla nähtiin myös joitain lisenssitekstipelejä, joista tunnetuimmat ovat The Hitchhiker’s Guide to the Galaxy ja The Hobbit. Linnunradan käsikirja liftareille -teoksen kirjoittaja Douglas Adams osallistui myös itse pelin suunnittelemiseen sekä kirjoitti Infocomille toisenkin pelin nimeltään Bureaucracy. Vuonna 1982 julkaistu The Hobbit oli yksi aikansa menestyksiä ja myi yli 100 000 kappaletta. The Hobbitin parseri oli aikaansa nähden erittäin kehittynyt. Kahden sanan komentojen sijaan pelin Inglish-parseri ymmärsi monimutkaisia lauseita, adverbeja ja välimerkkejä, mikä mahdollisti ennennäkemättömän luonnolliset komennot.

Eräs tekstipelihistorian virstanpylväistä oli Infocomin julkaisema ”seksifarssi” Leather Goddesses of Phobos, joka aiheutti aikanaan niin paheksuntaa kuin hihittelyä. Kyseessä oli humoristinen ja eroottissävytteinen peli, joka tarjosi mahdollisuuden pelata niin mies- kuin naishahmoa ja valita pelin rivousasteen.

Graafisten pelien voittokulku

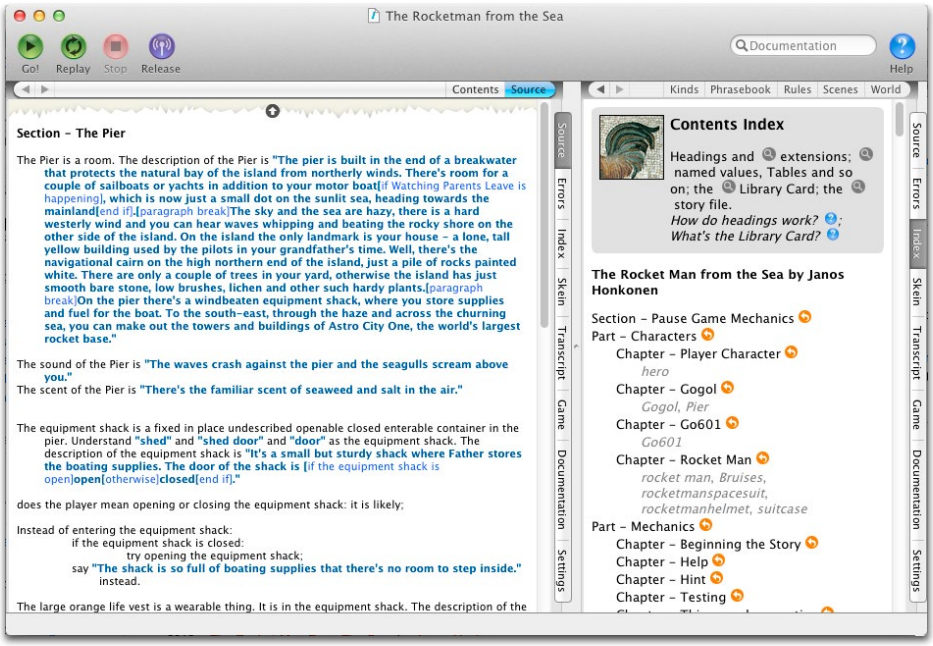
Tekstipelien kaupallinen suosio hiipui 90-luvun vaihdetta lähestyttäessä. Vuonna 1986 julkaistu Space Quest, 1987 päivänvalon nähnyt Leisure Suit Larry ja vastaavat seikkailupelien klassikot olivat näyttäviä ja helpommin lähestyttäviä kuin puhtaat tekstipelit.

Vanhat seikkailupelitalot eivät sopeutuneet uuteen aikakauteen. Niinpä Activision osti Infocomin vuonna 1986, Level 9 Computing lopetti toimintansa 1991 ja Magnetic Scrolls laittoi lipun luukulle 1992. Infocomin seuraaja Legend Entertainment julkaisi 90-luvun alussa muutamia graafisia tekstipelejä, kuten Gateway-pelit ja kolme humoristista Spellcasting-seikkailua, mutta vuodesta 1993 myös Legend julkaisi vain graafisia seikkailupelejä.

Tekstipelien monet nimet

Suomalaisittain tekstipeliksi kutsutaan peliä, jossa pääpaino on tekstimuotoisilla kuvailuilla ja komennoilla, mutta mukana voi olla myös animoituja still-kuvia. Kyseessä on seikkailupeli, jos pelissä on kehittyneempää animaatiota, jota voi ohjailla esimerkiksi liikuttamalla hahmoa ruudussa, kuten ensimmäisissä Leisure Suit Larry -peleissä. Termillä interaktiivinen fiktio viitataan pääasiassa uuden polven tekstipeleihin.

Englanninkielisessä maailmassa nimityskäytäntö on hieman sekavampi. Termi interactive fiction viittaa pääasiassa tekstipeleihin, mutta sitä voidaan myös käyttää seikkailupeleistä. Jotkut tekevät myös eron text adventuren ja interactive fictionin välillä. Tässä ensimmäisellä tarkoitetaan kevyitä puzzle-pelejä, jälkimmäisellä kirjallisempia, kerrontaan painottuvia teoksia.



Inform 7 -kehitysympäristö mahdollistaa koodaamisen lähes luonnollisella kielellä.

Tähän loppui käytännössä tekstipelien historia kaupallisina tuotteina ja pelien kehittäminen siirtyi harrastajatoiminnaksi.

Infocomin perintö ja TADS

Vaikka graafiset pelit valtasivat täysin kaupalliset markkinat, tekstipelit jatkoivat elämäänsä tiiviin harrastajapiirin ansiosta. Eräs merkittävä tekijä tälle oli Infocomin nerokas ratkaisu olla kääntämättä pelejään suoraan natiivikoodiksi, joka toimii vain tietyllä käyttöjärjestelmällä tai alustalla. Sen sijaan pelit käännettiin Z-Code-välikoodiksi, joka tarvitsee Z-machine-tulkin toimiakseen. Tämä tarkoittaa sitä, että pelejä voi pelata millä tahansa alustalla, jolle joku tekee toimivan tulkin.

InfoTaskForce-niminen harrastajaryhmä purki Infocomin Z-Code-koodin jo vuonna 1987 ja teki oman avoimen lähdekoodin ”soittimensa”, joka mahdollisti Infocomin pelien pelaamisen monilla uusilla alustoilla. Näin koko pelivalikoima saatiin tuotua uusilla alustoille kertarysäyksellä, eikä niitä tarvinnut kääntää yksitellen. Yksinkertaistaen tätä voi ajatella niin, että välikoodiksi käännetty peli on kuin mp3-musiikkitiedosto, jonka voi toistaa millä tahansa laitteella, jolle on kehitetty mp3-soitto-ohjelma.

Tekstipelien harrastajajulkaisussa tärkeässä osassa ovat Michael J. Robertsin vuonna 1988 julkaisema TADS, Text Adventure Development System, sekä Graham Nelsonin vuonna 1993 julkaisema Inform, jolla oli mahdollista luoda Infocomin käyttämää Z-Code-koodia.

Interactive Fiction Competition, tuttavallisemmin IFcomp, järjestetään jo 19. kerran. Kilpailun voittajat ovat hyvä tapa tutustua tekstipelikentän parhaimmistoon.

Seikkailusta kirjallisuuteen – harrastajan näkökulma

Oma historiani tekstipelien harrastajana alkaa jo 80-luvulta, jolloin pelejä tuli pelattua ala- ja yläaste-englannilla sanakirja toisessa kädessä. Saan kiittää englannin kielen kympeistäni niitä tunteja, jotka vietin The Hobbitin, Castle Adventuren, Fools Goldin, Jinxterin, Aztec Tombin ja vastaavien pelien parissa.

Suurin osa kohtaamistani peleistä oli suhteellisen yksinkertaisia seikkailuja. Koska luin samaan aikaan paljon tieteis- ja muuta kirjallisuutta, muistan kevyen puzzle-painotteisuuden ärsyttäneen minua jo tuolloin. Nämä pelit voisivat olla niin paljon enemmän! Väsäilin itse joitain pelintynkiä, mutta koska en ollut silloin kuullut Informin tai TADSin kaltaisista järjestelmistä, parserin koodaaminen täysin nollasta osoittautui aina pullonkaulaksi.

Tekstipeliharrastus jäi hiljaiseloon kymmeneksi vuodeksi, kunnes 2000-luvun alussa törmäsin sattumalta verkossa Photopia-nimiseen peliin. Pieni nostalgian läikähdys sai minut lataamaan pelin ja kokeilemaan, millaista kamaa genressä onkaan luotu vuonna 1998.

Peli jätti hiljaiseksi, todella hyvällä tavalla. Olin tietämättäni ladannut yhden lajityypin uusista klassikoista, jossa oli painotettu puzzlejen sijaan kirjallista sisältöä ja kokemuksellisuutta. Photopia on erinomainen esimerkki uuden aikakauden tekstipeleistä, joilla ei käsikirjoituksen ja tarinankerronnan tasolla ole hävettävää ”oikean” kirjallisuuden rinnalla.

Harrastajat luovat yhä keveitä seikkailupelejä, mutta myös tiukkaa proosaa, jossa fyysisen maailman esinepähkinöitä tärkeämpää saattaa olla hahmojen välinen kanssakäynti tai yleinen kirjallinen kokemus. Eräs toinen merkittävä klassikko on Emily Shortin Galatea, joka koostuu kokonaan keskustelusta päähenkilön ja elävän patsaan välillä. Keskustelun kulusta riippuen tarina voi päättyä hyvin eri tavoilla.

Kilpailuja, palkintoja ja aktiivinen yhteisö

Tekstipelien ympärille on muodostunut aktiivinen ja innokas faniyhteisö, joka on toiminut jo 80-luvulta saakka. Alun perin harrastajat kokoontuivat ja keskustelivat vuonna 1987 perustetuissa rec.arts.int.fiction- ja rec.games.int-fiction-uutisryhmissä, joista keskustelu on vasta viime vuosien aikana siirtynyt modernimmalle nettifoorumille, osoitteeseen intfiction.com/forum.

Vuosittain julkaistaan kymmeniä uusia pelejä, jotka ovat käytännössä kautta linjan ilmaisia. Julkaisusesonki painottuu yleensä loppuvuoteen, jolloin järjestetään tekstipelien suurin vuosittainen kilpailu IFcomp. Sitä tasapainottamaan perustettiin toinen kilpailu nimeltään Spring Thing, jolla haluttiin houkutella uusia julkaisuja myös kevätkaudelle. Edellisten lisäksi vuosittain jaetaan XYZZY-palkinnot eri kategorioissa, kuten paras peli, parhaat pelihahmot, paras puzzle ja niin edelleen.

Harrastajakentän merkittäviin nimiin kuuluvat mm. Andrew Plotkin, Emily Short, Adam Cadre sekä Graham Nelson. Plotkin pääsi otsikoihin onnistuttuaan keräämään Kickstarter-kampanjalla 31 000 dollaria Hadean Lands -tekstipelin kehittämiseen. Cadren Photopia tuuppasi tekstipelikenttää uuteen kirjallisempaan suuntaan, Graham Nelson on Inform-

Tekstipelitulkit

Tekstipelien pelaamiseen tarvitaan tulkki, eräänlainen pelitiedostojen ”soitin”.

- Gargoyle (OS X, Windows, Linux): http://ccxvii.net/gargoyle/
- Zoom (OS X, Unix, Palm OS, RISC OS): http://www.logicalshift.co.uk/unix/zoom/
- Windows Frotz 2002 (Windows):
- Frotz (iOS): https://itunes.apple.com/fi/app/frotz/id287653015
- JFrotz (Android): https://play.google.com/store/apps/details?id=com.whj1972.jfrotz
- Lisätietoja tiedostomuodoista ja tulkeista löytyy osoitteesta http://ifwiki.org/index.php/Interpreter

järjestelmän pääkehittäjä ja Emily Short on paitsi luonut useita palkittuja pelejä, myös osallistunut aktiivisesti Inform-järjestelmän kehittämiseen.

Kuinka päästä alkuun interaktiivisen fiktion pelaamisessa?

Moderneja tekstipelejä ei julkaista itsenäisinä, ajettavina peleinä, vaan niiden pelaamiseen tarvitaan tulkkiohjelmisto eli tekstipelien ”soitin”. Hyviä vaihtoehtoja ovat Windows-, OS X- ja Linux-alustoille löytyvä Gargoyle ja OS X:lle julkaistu helppokäyttöinen Zoom. Pelaaminen onnistuu myös iOS- ja Android-laitteilla jo jopa joillakin graafisilla laskimilla.

Suurin osa moderneista sekä vanhoista ei-kaupallisista tekstipeleistä on ladattavissa The Interactive Fiction Databasesta, joka sijaitsee osoitteessa http://ifdb.tads.org. OS X:n Zoom mahdollistaa myös pelien etsimisen ja lataamisen suoraan tulkkiin ”Find more” -nappulaa painamalla.

zblorb, z5, z8, gblorb, blb, acd, gam...

Tekstipelejä on luotu 20 vuotta harrastajilta harrastajille, joten valitettavasti tiedostomuotojen selkeys ja käytettävyys jättävät toivomisen varaa. Pelejä on julkaistu lähes 20:ssä eri tiedostomuodossa, mutta osa niistä on hyvin harvinaisia.

Tällä hetkellä pelejä luodaan pääasiassa kolmella järjestelmällä: Informilla, TADSilla sekä Hugolla. Suosituin näistä on Inform, joka luo Infocomin Z-machinen käyttämää Z-Codea sekä uudempia gluxl-tiedostoja. Kakkossijalla on TADS. Hugolla ja muilla järjestelmillä tehdyt pelit ovat suhteellisen harvinaisia.

Tarinat julkaistaan usein blorb-wrapperissa, jolloin niiden tiedostopääte on .zblorb tai .gblorb. Wrapperit mahdollistavat multimedian lisäämisen pakettiin, ja tiedostopäätteen ensimmäinen kirjain kertoo, onko kyseessä Z-Machine- vai gluxl-muotoinen peli. Ilman wrapperia Z-Code-tiedostojen pääte on .z5 tai .z8, gluxl-pelien .ulx. TADS-tiedostojen pääte on .gam tai .t3, Hugo-tiedostojen .hex.

Edellinen oli raaka yksinkertaistus alaa vaivaavasta tiedostomuoto- ja nimikaaoksesta, sillä variaatioita ja julkaisujärjestelmiä on moninkertaisesti enemmän. Sotkun vuoksi uuden seikkailijan kannattaa ehdottomasti kokeilla tekstipelejä tulkilla, joka tukee suoraan useimpia tekstipelien tarinatiedostomuotoja. Suositeltavia tulkkeja ovat muun muassa Gargoyle ja Zoom.

Oma peliprojekti käyntiin

Ennemmin tai myöhemmin harrastaja miettii oman pelin tekemistä. Tällä hetkellä ylivoimaisesti käytetyin järjestelmä on Inform 7 (http://inform7.com/), jonka lähestymistapa pelikehitykseen on vallankumouksellinen. Aiemmat versiot sekä Informista että TADSista ovat melko perinteistä koodia hakasulkeineen, muuttujineen ja funktioineen, mutta Inform 7:n kehittäjä Graham Nelson otti järjestelmän uusimman version kehittämisessä rohkean askeleen: jos pelit perustuvat luonnolliseen kieleen, myös niiden koodaamisen pitäisi onnistua luonnollisella kielellä.

Käytännössä Inform 7:n koodi voi näyttää esimerkiksi seuraavalta: ’The cottage is a room. The coat is a wearable thing in the cottage. Instead of taking the coat, say ”It is not your coat to take.”’

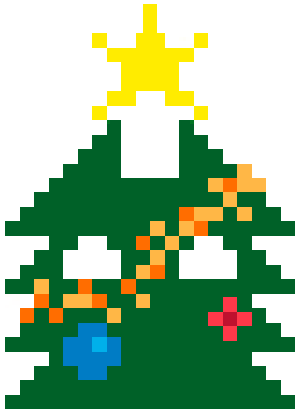
Inform 7:n luonnollisen kielen koodaus ei ole pelkkä kikka, vaan pienen totuttelun jälkeen se on suorastaan hämmentävän toimiva. Järjestelmä ei käännä luonnollista kieltä suoraan välikoodiksi, vaan ensin perinteisemmäksi Inform 6 -koodiksi, joten se on rakennettu toimivaksi todetulle pohjalle. Myös itse kehitysympäristö on selkeä ja helppokäyttöinen niillekin, jotka eivät ole aiemmin ohjelmoineet.

TADS3 (http://www.tads.org) puolestaan tarjoaa perinteistä ohjelmointia, jonka parissa kokeneet koodaajat ovat kuin kotonaan. Sen varsinainen kehitysympäristö on saatavilla vain Windows-järjestelmille, joten esimerkiksi OS X -käyttäjien on tyytyminen komentotulkkityökaluihin. TADS3 on olio-ohjelmointikieli, siinä missä Inform 7 on sääntöpohjainen järjestelmä. TADS3 tarjoaa myös melko monimutkaisen maailmamallin, Inform 7 taas minimaalisen. Aloittelijan näkökulmasta Inform 7 mahdollistaa yksinkertaisen pelin kirjoittamisen yhdessä illassa ilman mitään ohjelmointikokemusta, TADS3 taas vaatii enemmän perehtymistä. Tarkempi erittely järjestelmien eroavaisuuksista löytyy Brass Lantern -sivustolta: http://brasslantern.org/writers/iftheory/tads3andi7.html

Vähemmän käytetty Hugo-järjestelmä on tutustumisen arvoinen niille, jotka haluavat luoda paljon multimediaa sisältäviä pelejä.

Laadukkaan pelinkirjoituksen ystäville

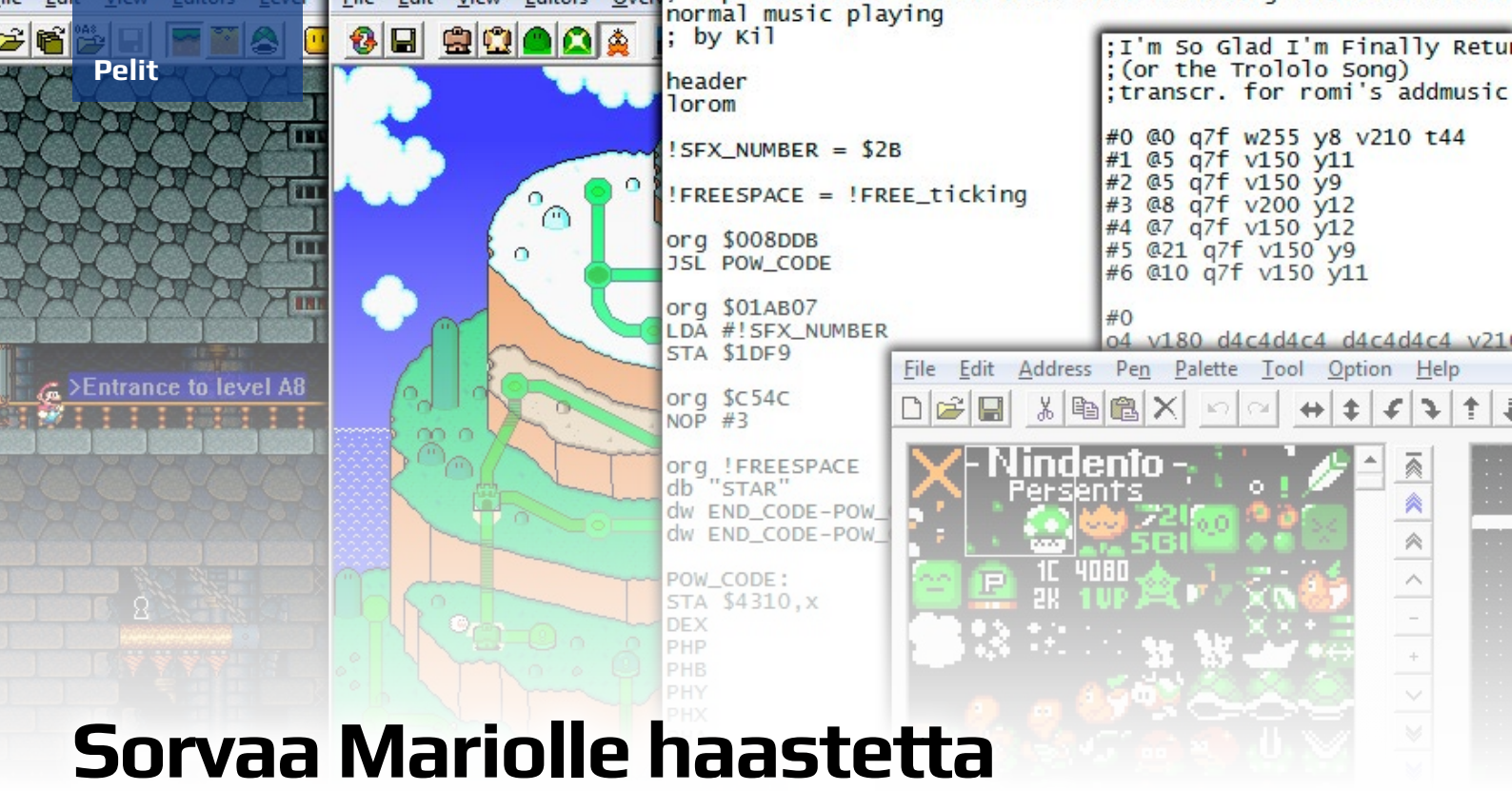
Tekstipelit tulevat tuskin nousemaan enää kaupallisesti merkittäväksi tuotteeksi, mutta taiteellisessa mielessä ne ovat erittäin kiinnostava pelityyppi. Epäkaupallinen kehitystyö antaa tekijöille mahdollisuudet kirjoittaa kokeilevia, rankkoja ja haastavia tarinoita, jotka eivät läpäisisi kaupallisten julkaisujen seulaa. Joukossa on paljon huteja, mutta vähintään yhtä paljon täysosumia, jotka antavat osviittaa siihen, millaista pelikäsikirjoittaminen voisi olla parhaimmillaan myös kaupallisella puolella.



Modernien tekstipelien valioita – kirjoittajan suosikit

- Photopia (Adam Cadre, 1998): Tarina, joka yhdistää arkisen elämän tragedian nuoren tytön unelmiin Mars-lennosta.
- Galatea (Emily Short, 2000): Keskustelu taideteoksen, elävän patsaan, kanssa.
- Vespers (Jason Devlin, 2005): Rutto niittää kuolemaa luostarin ulkopuolella ja jokin muu sen sisällä.
- The Warbler’s Nest (Jason McIntosh, 2010): Nuori nainen seisoo joen varrella ruovikossa. Se jokin odottaa mökissä.
- Violet (Jeremy Freese, 2008): Istu työhuoneessasi, kirjoita tuhat sanaa opinnäytetyötäsi äläkä anna minkään – minkään – häiritä.

Lisää laatupelejä kannattaa etsiä vuosittaisen IF-comp- (http://ifcomp.org) ja Spring Thing (http://www.springthing.net/) -kilpailun voittajista sekä ZYXXY-palkinnon saaneiden joukosta (http://xyzzyawards.org).



Sorvaa Mariolle haastetta

Kuka ei olisi haaveillut omien Super Mario -kenttien väsäämisestä? Nykyään se on mahdollista. Aikanaan ruutuvihkoon tuherretut suunnitelmat voi lopultakin herättää henkiin.

Teksti: Pauli Marttinen

Super Mario World on todellinen kuolematon klassikko. Vuonna 1990 julkaistun pelin loputonta suosiota syventää entisestään *romhack*-harrastus eli rom-moduulin sisällön muokkaaminen. Tietokoneelle tallennettua Super Nintendon pelimoduulin kuvatiedostoa eli smc-tiedostoa (Super Mario Cartridge) voi käsitellä niin paljon kuin mielikuvitusta riittää. Valmiiden työkalujen avulla peliin voi tehdä omien kenttien lisäksi omat kartat, oman musiikin, omaa grafiikkaa, omia vihollisia sekä bugien korjauksia ja uusia ominaisuuksia, kuten dialogeja.

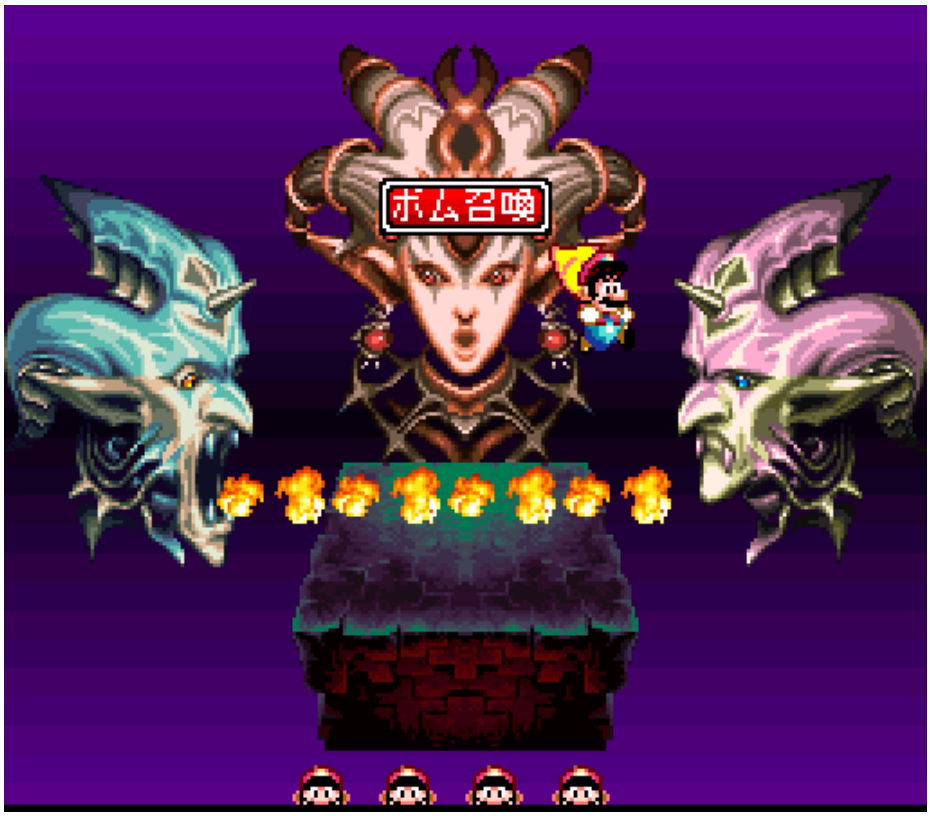
Pelin hakkeroinnille on monia syitä. Yleisin lienee omien kenttien tekeminen valmiiseen peliin. Jotkut taas tahtovat tehdä uuden pelin vanhan pelin pohjalta, ja joskus ajatuksena on vain nostaa valmiin pelin vaikeusastetta. Joissakin tapauksissa on myös kyse pelkästä fanikäännöksen tekemisestä.

Yleisimmin Super Mario Worldiin yhdistetään elementtejä muista peleistä. Suosituimpia lähteitä ovat Donkey Kong, Mega Man, Touhou Project ja muut Super Mario -pelit. Ahkerat ja avuliaat harrastajat ovat muuntaneet ja valmistaneet näiden ja muiden pelien musiikkia, vihollisia, palikoita ja grafiikkaa muotoon, jossa niitä voi liittää omaan romhackiin.

Super Mario Worldin ohella suosittuja hakkeroinnin kohteita ovat muut Mario-pelit, Pokémonit, Mega Manit ja Final Fantasyt. Todennäköisesti juuri työkalujen ja nostalgian vuoksi Super Mario Worldista on tullut selvästi suosituin peli romhack-yhteisölle. Myös romhackien peluuvideot YouTubessa ovat suosittuja, yhtäältä hauskojen uusien ideoiden vuoksi ja toisaalta viihdyttävien kommentaarien ansiosta.

Oman romhackin tekeminen vaatii luonnollisesti reippaasti osaamista ja jaksamista, mutta valmiista kentistä pääsee nauttimaan melko pienellä vaivalla.

FPI: The Second Reality Project (2002) oli ensimmäisiä valmistuneita Super Mario Worldiin perustuvia romhackeja.



Carol: Brutal Mario on pirullisen vaikea ja rikkoo Mario-kaavaa mielenkiintoisilla omilla tempuillaan.

Merkittäviä romhackeja

Super Mario Worldin hakkerointi on 2000-luvun juttu. Nimimerkki FPI:n vuonna 2002 luoma The Second Reality Project on ensimmäisiä romhackeja. Se on niin sanottu vaniljahack, eli vain kenttiä on muutettu. Grafiikka, musiikki ja ohjelmointi on jätetty entiselleen. The Second Reality Project on haastavuudesta huolimatta jäänyt elämään yhtenä romhack-harrastuksen klassikoista. Siitä on vuosien saatossa tehty päivitettyjä versioita ja jatko-osia.

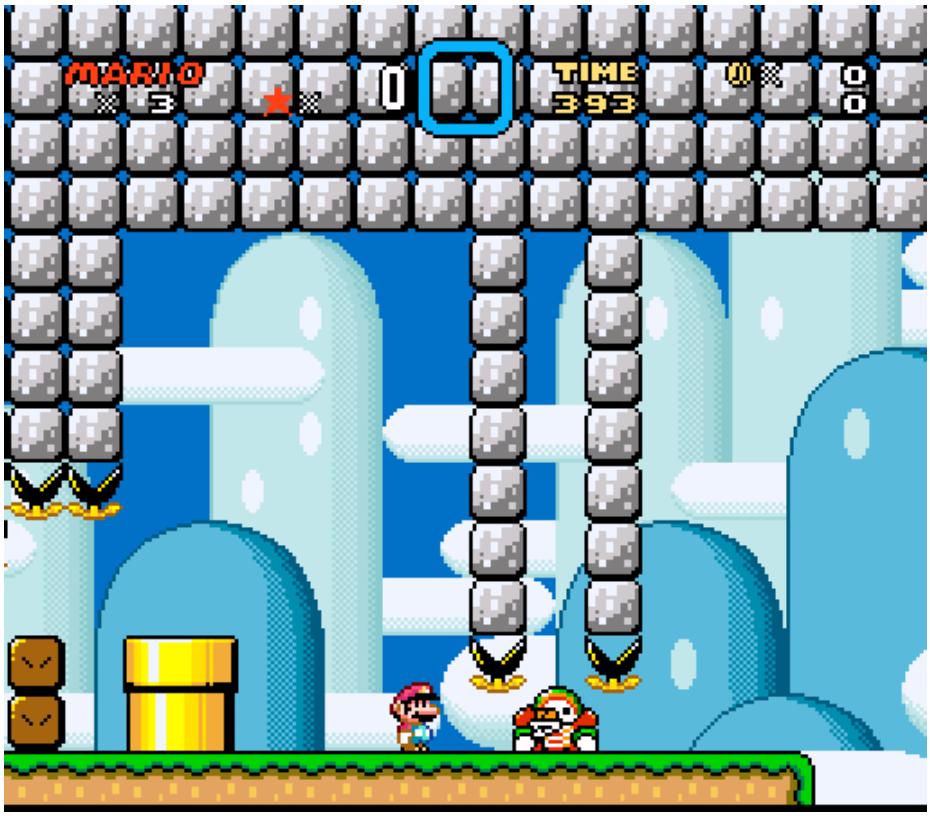
Myös Carolin keskeneräiseksi jäänyt Brutal Mario oli aikoinaan edistyksellinen. Se on osin aivan tyypillinen japanilainen romhack. Ensimmäiset kentät ovat hyvin helppoja ja yksinkertaisia, mutta vaikeustaso nousee nopeasti ja korkealle. Jokaisessa kentässä on oma erityinen jipponsa. Esimerkiksi seitsemännen maailman kummitustalossa on kummitus, jonka ei saa antaa nälkiintyä, tai se syö Marion. Brutal Marion merkittävyys on sekä kikoissa että linnojen loppuvastuksissa. Näin laajaa ohjelmoinnin ja grafiikan muokkausta ja lisäämistä ei ollut ennen tehty. Pelin kehityksestä ei ole kuulunut mitään vuoden 2008 jälkeen, mutta demoversiota voi pelata varsin pitkälle.

Brutal Marion tavoin keskeneräiseksi jäänyt S.N.N.:n Mario's Keytastrophe vuosilta 2007–2009 on ensimmäinen musiikin korvannut romhack. Myös aiheensa puolesta peli pyörii musiikin ympärillä. Maailmoja ovat muun muassa Legato Plateau ja Andante Region sekä kenttiä esimerkiksi Trumpeteer Cliffs ja Blue Melody Palace. Kentissä tehtävänä on löytää avain. Musiikki oli viimeinen hakkeroimaton Super Mario Worldin osa, ja Mario's Keytastrophen käyttämän *addmusic*-työkalun myötä sekin on saatu hallintaan.

S.N.N.: Mario's Keytastrophe (2007–2009) oli ensimmäinen pelin ääniraitaa muokannut teos.

Yksi pahamaineisimmista romhackeista on japanilaisen T. Takemoton Yuujin Mario ('*kaveri-Mario*'), joka lännessä tunnetaan nimellä Kaizou Mario ('*muokattu Mario*'). Se alkoi kerätä suosiota, kun japanilaiselle videosivustolle, Nico Nico Dougalle, ladattiin romhackista läpipeluuvideosarja. Kärsivällinen pelaaja R. Kiba päihittää armottoman vaikean pelin apunaan vain loputtomat elämät. Sama videosarja on myös YouTubessa otsikolla Asshole Mario. Nykyään epäreilun vaikeiksi tehtyjä romhackeja verrataan ensimmäisenä juuri tähän ja tietyntyyppisiä esteitä sanotaan kaizou-esteiksi.

Sooloteosten lisäksi romhackeja tekevät myös useat isommat yhteisöt. Massiivisimpiin projekteihin lukeutuvat muun muassa SMW Central -sivuston An SMW Central Production ja Talkhaus-yhteisön A Super Mario Thing. Näille yhteistä on pelin poikkeuksellisen monipuolinen muokkaaminen eri alueiden osaajien ansiosta.



T. Takemoto: Yuujin/Kaizou Mario (2007) on hyvä vaihtoehto, jos haluaa kirota myöhään yöhön epäreilulle suunnittelulle – tämä nimittäin on kertaluokkaa vaikeampi kuin alkuperäinen peli.

Työkalut kokeilijalle

Romhackeja jaetaan netissä ips-tiedostoina (international patching system). Ips-tiedostoon tallennetaan pelkästään valmiin romhackin erot alkuperäiseen smc-tiedostoon verrattuna. *Floating IPS* -työkalulla yhdistetään ips- ja smc-tiedostot. Syntyvän kuvatiedoston voi ajaa emulaattorilla, esimerkiksi ZSNES:llä. Ips-tiedostoja käyttämällä vältytään Super Mario Worldin itsensä laittomalta jakamiselta.

Ips- ja smc-tiedostot yhdistetään Floating IPS:n *Apply Patch* -painikkeella. Toimenpiteestä syntyvän tiedoston voi ajaa emulaattorilla aivan kuin minkä tahansa tavallisenkin pelin. Romhack taas tallennetaan IPS-tiedostoksi *Create Patch* -painikkeella. Työkalulle annetaan alkuperäinen kuvatiedosto, sekä romhackattu versio. Syntyvän tiedoston voi jakaa turvallisesti netissä.

Super Mario Worldista on liikkeellä erilaisia kuvatiedostoja, kuten englannin- ja japaninkielinen versio, mutta käytännössä kaikki ips-tiedostot pohjautuvat kuvatiedostoon nimeltä *Super Mario World (U) [!].smc*.

Lunar Magicillä alkuun

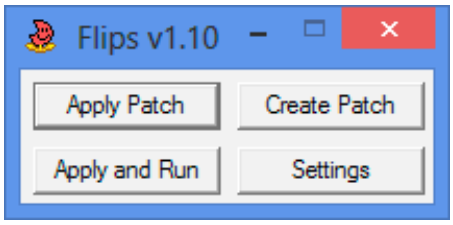
Lunar Magic on edistyksellisin editori Super Mario Worldiin. Kentät rakennetaan Lunar Magicilla, joten se on keskeisin ohjelmisto pelin muokkaamisessa. Teit omat resurssit millä halusit, Lunar Magicilla koostat niistä pelattavaa.

Lunar Magic avaa ja tallentaa kentät suoraan smc-tiedostoon. Perusnäkymässä on kolme varsinaista muokkaustilaa: Layer 1, Layer 2 ja Sprite. Layer 1 sisältää maa-, vesi-, ja muita palikoita, joiden kanssa Mario pääasiassa vuorovaikuttaa. Layer 2 sisältää samoja palikoita, ja se voi liikkua suhteessa Layer 1:een. Sprite-kerros sisältää luonnollisesti spritet. Taustakuvaa (*background*) ja Layer 2:ta ei voi olla kentässä samanaikaisesti, joten jos kentässä on taustakuva, Layer 2:ta ei voi editoida.

Add Sprites -ikkunassa on lueteltuna spritejä, joihin lukeutuu vihollisten ja esineiden lisäksi komentoja, kuten automaattisen skrollauksen ohjaimet. Monien spritejen grafiikka on riippuvaista tiilisetistä (*tileset specific sprites*) eli siitä, mitä grafiikkaa kentän käyttöön ladataan. Mikäli näiden spritejen oikea grafiikka sijaitsee muualla kuin ladattavissa osissa, ne näyttävät rikkinäisiltä niin editorissa kuin pelissäkin.

Add Objects -ikkunassa on lueteltuna Layer 1:lle ja 2:lle lisättäviä palikoita. Myös monien palikoiden grafiikka on riippuvaista käytetystä tiilisetistä. Spritet ja objektit lisätään kenttään valitsemalla ne luettelosta ja painamalla hiiren kakkosnapilla kenttää pääikkunassa. Objektien kokoa voi kappaleesta riippuen venytellä tarpeen mukaiseksi.

Vaikka Marion pääsisääntulo ja puolivälisisääntulo kenttään eivät varsinaisesti ole spritejä, niitä voi siirtää sprite-muokkaustilassa samoin kuin muita spritejä.



Floating IPS -työkalulla yhdistetään romhackin muutokset alkuperäiseen smc-tiedostoon.

Kentät jakautuvat pienempiin yksiköihin eli screeneihin. Yksiköiden rajat saa näkyviin F1-näppäimellä. Kentästä toiseen vievät ovet ja putket toimivat screen exit -määreellä. *Add/Modify/Delete Screen Exits* -ikkunassa määritetään, mihin kenttään ovet ja exit enabled -putket vievät. Jos poistumismäärä on kenttänumero eikä toissijainen poistumisreitti eli secondary exit, Mario astuu uuteen kenttään sen pääsisääntulossa.

Toissijaisilla poistumisreiteillä Mario astuu uuteen kenttään muualla kuin pääsisääntulon sijainnissa. Ne määritellään *Modify Secondary Entrances* -ikkunassa. *Secondary Entrance/Exit Number* -luettelosta valitaan määriteltävä poistumisreitti ja sen sijainti määritellään samoin kuin pääsisääntulon sijainti.

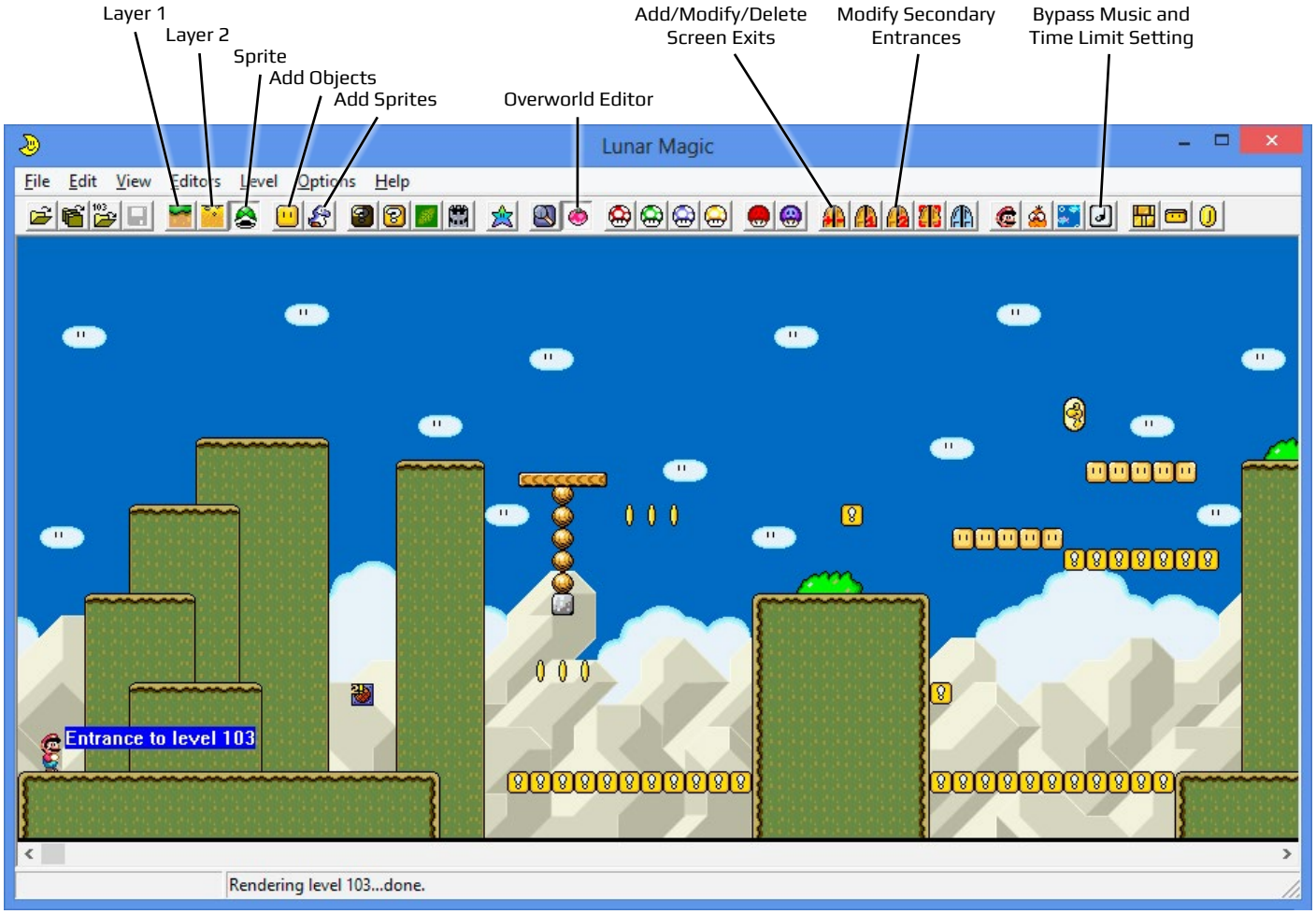
Kartan editointi *Overworld Editorissa* on oma taiteenlajinsa, jota varten on käytännössä pakko tutustua syvällisempiin oppaisiin. Karttaeditorissa kannattaa ottaa kenttänumerot näkyviin F1-näppäimellä nähdäkseen, minkä kenttien editoinnista on hyvä lähteä liikkeelle.

Musiikit vaihtoon

Suositellut työkalut musiikin lisäämiseen ovat *HuFlungDu's Addmusic* ja *AddmusicM*. Ensimmäinen on ominaisuuksiltaan ja käytöltään yksinkertainen, jälkimmäinen monipuolisempi mutta myös monimutkaisempi. Lyhyenä esimerkkinä lisäämme tässä pätkän tuttua laulua sekä kartta- että kenttämusiikiksi HuFlungDu's Addmusicia käyttäen.

Musiikin lisäämistä varten smc-tiedostoa on ensin laajennettava. Laajennus tapahtuu Lunar Magicin valikosta *File > Expand ROM*. Sen jälkeen kopioidaan Addmusicin hakemistossa sijaitsevat *level-*, *ow-* ja *misc*-hakemistot sekä *AddMusic.ini* siihen hakemistoon, jossa smc-tiedosto on. Level-hakemistoon sijoitetaan kenttämusiikkitiedostot ja ow-hakemistoon karttamusiikkitiedostot. Misc sisältää sekalaisia ääniä, kuten kuolemismusiikin.

Kopioi oheinen listaus ja tallenna se tekstitiedostona level- ja ow-hakemistoi-



Super Mario World -romhackin tekijälle paras työkalu on monipuolinen Lunar Magic 2.20.

hin. Tiedoston nimi voi olla esimerkiksi *kappale.txt*. Avaa AddMusic.ini ja lisää kappaleen tiedostonimi *[overworld]*- ja *[level]*-osuuksien mihin tahansa kenttään.

Addmusic on komentoriviohjelma, joten siirry komentorivillä smc-tiedoston hakemistoon ja aja ohjelma näin:

```
> [addmusicin sijainti]\AddMusic.exe
romhack.smc -i AddMusic.ini
```

Mikäli musiikin lisääminen onnistuu, kappaleen voi valita kenttään Lunar Magicin *Bypass Music and Time Limit Setting* -ikkunassa ja karttaan *Change Submap Music Selection* -ikkunassa.

Erillinen *AM4 Player* -ohjelma toistaa musiikkia suoraan tekstitiedostosta. Kappaletta voi siis myös kuunnella lisäämättä sitä peliin.

Kun lisäät dataa Super Mario Worldiin, muista pitää smc-tiedostosta varmuuskopiota. Virheellisen tai toisille työkaluille tarkoitetun datan lisääminen saattaa hajottaa koko tiedoston.

Pintaa syvemmälle?

Blocktool Super Deluxe on suositeltu työkalu uusien palikoiden lisäämiseen. *Romi's Spritetool* ja *Tessera* ovat suositellut työkalut spritejen lisäämiseen. Spritetoolin ja Tesseran mukana tulee valmiiksi suuri valikoima lisättäviä spritejä. Blocktoolin ja Spritetoolin mukana tulevan *xkas*-ristiassemblerin avulla voi myös koodata omia. *YY-CHR* on suosittu työkalu grafiikan muokkaamiseen. Sillä voi myös piirtää omaa grafiikkaa, jota voi lisätä peliin Lunar Magicilla. Näihin ja moniin muihin työkaluihin saa haettua lisää resursseja SMW Central -sivustolta.

Super Mario Worldin muokkaamisessa täytyy muistaa, että kaikki työkalut ovat harrastajahakkereiden valmistamia, eikä peliä ole suunniteltu muokattavaksi ja laajennettavaksi. Tämän vuoksi jotkut muokkaustoimenpiteet voivat tuntua työläiltä ja epäintuitiivisilta. Kaikki on kuitenkin mahdollista, minkä todistavat SMW Centralista ladattavat lukuisat mieltäräjäyttävät romhackit.

Työkalut kehittyvät, ja monet ovat avoimia lähdekoodiltaan, joten niiden kehittämiseen voi osallistua itsekin. Romhakkerointi on yhteisöllinen harrastus, ja uudet tekijät otetaan ilomielin vastaan. Ei muuta kuin vanhat ruutuvihot auki ja päästämään sisäinen kenttäsuunnittelija valloilleen!

Musiikkilistaus

```
#0 @1 w255 y11 t40
#1 @8 y7

#0
o4 g8 / > c4 < g8^16 a16
b4 e8^16 e16 a4 g8^16 f16
g4 c8^16 c16 d4 d8^16 e16
f4 f8^16 g16 a4 b8^16 > c16
d4^8 < g8;

#1
o2 r8 / c4 c4 e4 e4 f4 f4 e4 e4
d4 d4 c4 c4 c8 > c8 < b8 a8 g4 >
g8 < g8;
```

Mario-hakkerin tärkeimmät linkit

Lunar Magic: http://fusoya.eludevisibility.org/
SMW Central: http://www.smwcentral.net/
SMWiki: http://www.smwiki.net/wiki/Main_Page
ZSNES: http://www.zsnes.com/

Litteän maailman kehityksestä

Näin syntyi Division Cell

Androidille, iOS:lle ja selaimille kehitetyn Division Cellin puzzlemekaniikka on kenties uniikki. Pelin kehittäjä kertoo rakkaan lapsensa syntytarinan.

Teksti: Visa-Valtteri Pimiä

Tätä tekstiä kirjoittaessani Division Cellin julkaisupäivä lähestyy. Oman kaupallisen pelin valmistaminen ja julkaisu muuttui vuoden urakalla unelmasta todellisuudeksi. Mutta miten tähän pisteeseen päästiin? Alussa oli tviitti.

Huomasin alkukeväästä 2013, että entinen työkaverini, Evgeni Gordejev, oli tehnyt pelin Googlen Android-käyttöjärjestelmälle. Opiskeluaikanani olimme Evgenin kanssa yhdessä töissä Ardites-nimisellä ohjelmistoalan yrityksellä. Kun näin Play Storen etusivulla Tupsu-nimisen pelin, muistui mieleeni tämä kekseliäs tamperelainen ohjelmoija vuosien takaa.

Muutaman hetken peliä testattuani löysin Evgenin Twitter-tilin (@gabumba) ja lähetin hänelle seuraavan viestin:

```
”@gabumba If you need any help with
future porting efforts of Tupsu I’d be
happy to help you in any way :). Awesome
game t: visy from Ardites”

@notyourusualcup – 10:25 AM – 23 Feb 13
```

Tästä kaikki alkoi, mutta aloitimme keskustelut Tupsun porttauksen sijaan mahdollisista uusista peliprojekteista. Pian Evgeni esittelikin kehittämänsä pelin prototyyppiä. Ensimmäinen pelaamani versio oli yksinkertaisen näköinen mutta viehättävä.

Varhaista kehitystä

Projekti lähti JavaScript-sivusta, jolla oli kolme neliötä, joiden reunoilla oli pisteitä. Hetken tällä protolla leikittyäni alkoi satunnainen molemminpuolinen ajatustenvaihto. Välillä kului viikkojakin muiden projektien parissa, mutta sitten palasimme jälleen Evgenin kanssa juttusille mahdollisista peliin liittyvistä konsepteista.

Kevätkuukausien saatteessa peruspelin rakenne alkoi hahmottua. Neliöiden lisäksi mukaan testattiin ja hyväksyttiin kolmiot ja piirakat. Päätimme yhdessä peruspelimoodiksi Puzzle Moden, jossa olisi Angry Birdsistä ja monesta muusta mobiilipelistä tuttu kolmen tähden läpäisymalli.

Lähdimme tutkimaan, voisiko peliin generoida pseudosatunnaisia tasoja. Proseduraalisten algoritmien piti myös tarkistaa, että luotu taso on läpäistävissä ja visuaalisesti miellyttävä. Alkoi työ kahden muun pelimoodin parissa, joissa hyödynnettäisiin automaattisesti luotuja tasoja. Syntyi Time Attack, jossa kilpaillaan kelloa vastaan ja jossa hyvin suoritetuista kentistä tippuu lisäaikaa. Teimme myös Endless Moden, jossa kenttien välissä kerrotaan joka kerta erilainen tarina.

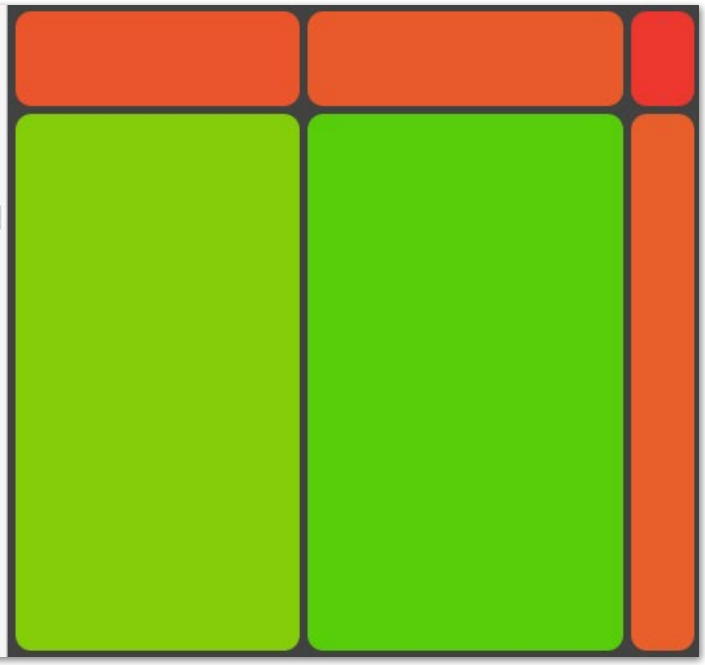
Huomasimme satunnaisluotuja tasoja testatessamme, että myös ohjelmallisesti luotu värivalikoima sopisi hyvin pelin käyttöliittymään. Valitsimme lopulta värit Flat UI -paletista, jossa on erityisesti korostettu käyttöliittymäelementtien kontrastia sekä pastellivärejä pelkistettyjen tyylien tarpeisiin.

Tärkeä yksityiskohta, jonka huomioimme jo varhaisessa vaiheessa kehitystä, oli pelin tasojen jakaminen muiden pelaajien kanssa. Tärkeimpänä vaikuttimena tähän oli oivallus pelitilanteen tallentamisesta pieneen määrään tietoa. Yksi suhteellisen lyhyt ja linkkiin mahtuva numerosarja riittää kertomaan tasogeneraattorille, luodaanko taso kolmioista, neliöistä vai piirakkamuodoista ja millainen tason muoto on.

Loppu on vain satunnaislukugeneraattorien iteroimista erilaisille tarkistuksille tason sisällön oikeellisuudesta, läpipelattavuudesta ja symmetrisyydestä. Näin pelin ei juurikaan tarvitse siirtää tietoa verkon yli. Linkissä mukana olevat tiedot riittävät kertomaan, kuka tason on jakanut, millä tuloksella ja millaiset numerot tuottavat juuri sen muotoisen ja värisen kuvion.

Tilatehokkuus ja tasojen muodostamat kuvioasetelmat tuovat mielleyhtymiä digitaalitaiteen pseudosatunnaisuuteen ja tilakoodaukseen ja toisaalta myös itämaisiin taidemuodon ja palapelin yhdistelmiin: origamiin ja tangramiin. Onpa joku myös verrannut peliä myös sudokuun ja sen numeroiden tasapainotukseen perustuvaan pelistrategiaan. Tietyssä mielessä samanlaisia menetelmiä käytetään myös Division Cellin palapelien ratkomiseen.

```
    rects.length = 0;
}

function generateLines(){
    linesV.length = 0;
    linesH.length = 0;

    level.boxW =
Math.round(settings.canvasSize/settings.gridStep/(level.
linesW+1))*(level.linesW+1)*settings.gridStep;
    level.boxH =
Math.round(settings.canvasSize/settings.gridStep/(level.
linesH+1))*(level.linesH+1)*settings.gridStep;

    for(var i = 0; i < level.linesW; i++){
        linesV.push(getExclusiveLine(linesV,
level.boxW));
    }
    for(var i = 0; i < level.linesH; i++){
        linesH.push(getExclusiveLine(linesH,
level.boxH));
    }

    function sortNumber(a,b) {
        return a - b;
```



Ensimmäinen prototyyppi ei vielä ollut kovin hiotun näköinen, mutta viehättävä yksinkertaisuus vetosi.

Tekniikasta ja julkaisusta

Käytämme pelissä PlayN-kirjastoa (https://code.google.com/p/playn/), jonka avulla kaikki pelilogiikka tarvitsi kirjoittaa Java-ohjelmointikielellä vain kerran. Se käännettiin Apachen Maven-työkalun ja Monotouchin avulla iOS:lle, Androidille ja HTML5-renderöijää käyttävälle JavaScriptille. Teknisten valintojen ansiosta voimme julkaista pelin kaikilla kolmella alustalla samanaikaisesti.

Pistelistat ja pelin sisäisten saavutusten seuranta piti toteuttaa kahdella eri tavalla: iOS:lla Game Centerillä ja Androidilla Google Play -palvelulla. Toiminnallisuus näissä kahdessa palvelussa on kuitenkin lähes identtinen, joten pystyimme vaivattomasti toteuttamaan kaikki ominaisuudet molemmilla isoilla mobiilialustoilla.

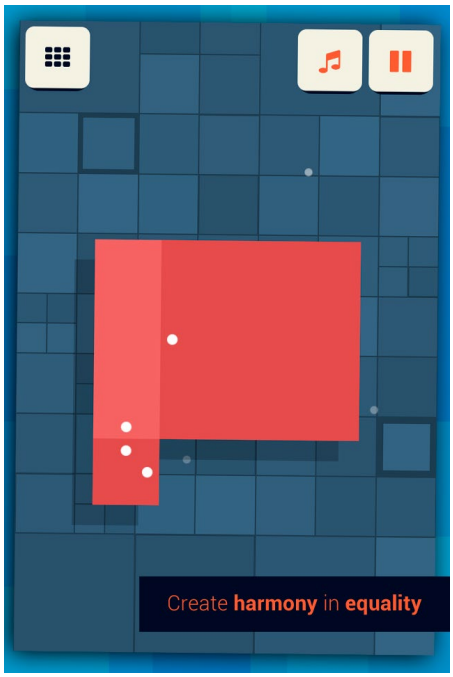
Julkaisu Applen ja Googlen alustoilla tapahtuu alustakohtaisten maksullisten ohjelmakauppojen kautta. Julkaisemisen ilosta livahtaa Applelle 99 taalaa vuodessa ja Googlelle 25 taalan kertamaksu.

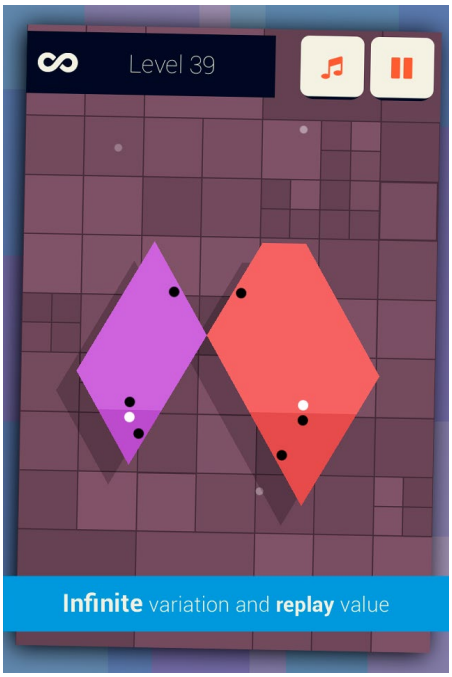
Itse julkaisemisessa hermoja raastavinta on Applen tarkastuksen läpäisy. Tarkasteluun lähetetty ohjelma saatetaan hylätä puutteellisena tai ohjelman tietojen virheellisyyden takia. Prosessin tekee kehittäjän kannalta hankalaksi sen ”musta laatikko” -tyylinen luonne. Tarkastuksesta ei saa monen päivän odottelun jälkeen kuin hyväksytyn tai hylätyn tuloksen.

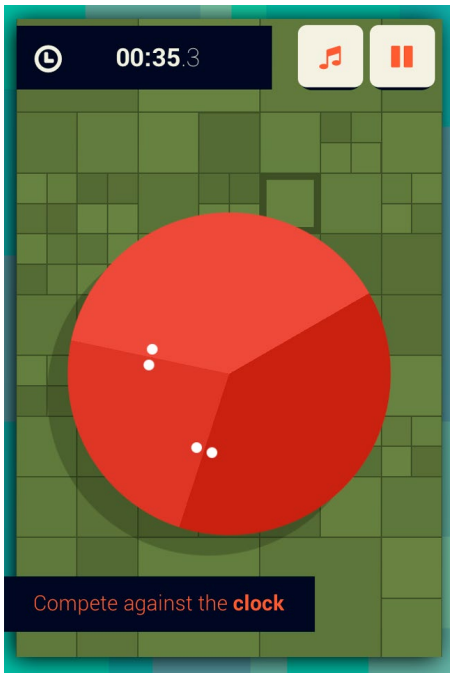
Ideasta kuvaksi

Division Cellissä on pääasiallisesti vektorigrafiikkaa. Pelkistetty ulkoasu sopii esitettäväksi enimmäkseen skaalautuvilla elementeillä kuten kolmioilla, neliöillä ja fonteilla. Nettisuunnittelijoille tuttu Font Awesome tulikin erittäin hyvään tarpeeseen nappeja ja ikoneita suunnitellessa, koska se sisältää suurimman osan tarvittavista käyttöliittymän symboleista.

Grafiikka on renderöity käyttäen OpenGL ES -kirjaston näytönohjaimen kiihdytystä. Se tekee pelistä keskivertopuzzlepeliä raskaamman, mutta mahdollistaa sulavat animaatiot ja kauniin pehmeäreunaiset geometriset kuviot myös iPadin valtavalla Retina Display -resoluutiolla. Suurin osa kohdelaitteista



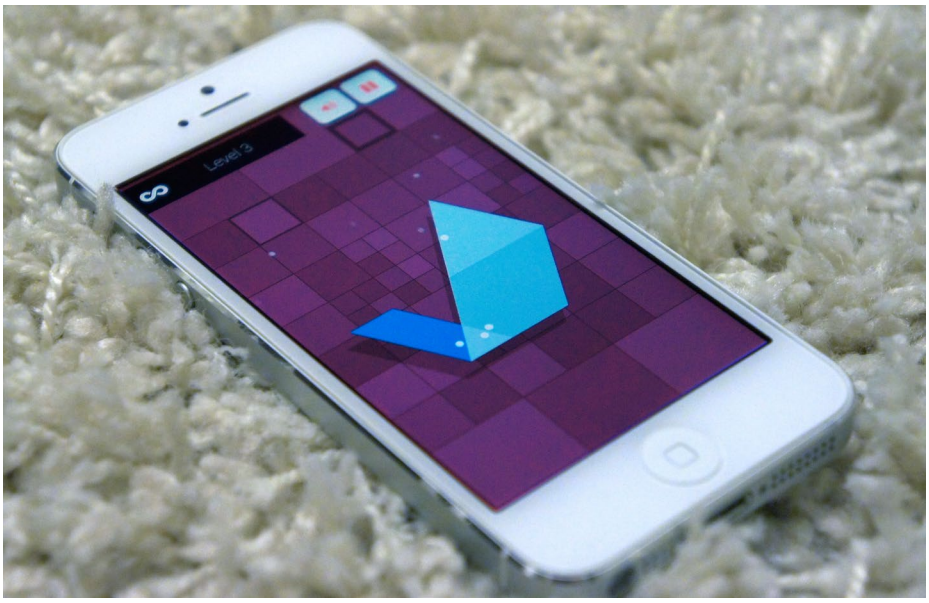




Pelin perusmuotoja ovat neliöt, kolmiot ja ympyrät. Kun muotoa hipaisee, kaikki sivut, joilla on valkoinen ympyrä, laajenevat. Mustasivuiset reunat taas supistuvat. Tavoitteena on saada kuvioiden kaikista osista samankokoisia.

on onneksi riittävän tehokkaita grafiikkasuorittimiltaan, joten peli toimii vähintäänkin tyydyttävästi, eikä pelaamisen ilo pilaannu karvalakkimallien omistajiltakaan.

Pelin Story Teller -niminen maskottihahmo animoitiin Flumpin avulla (http://threerings.github.io/flump/). Flump mahdollistaa monimutkaisten hahmoanimaatioiden purkamisen pala-animaatioiksi. Kössi Kengurun tai South Parkin tyyliin koodi asettelee palat päällekkäin ja liikuttelee niitä animaatiodatan määrittämässä tahdissa. Näin säästyy tilaa, kun animaation jokaista kuvaa ei tarvitse erikseen tallentaa kuvatiedostoksi – pelkkä palojen tallentaminen riittää.



Lopussa kiitos seisoo. Oma peli puhelimessa kauniina ja pelikelpoisena. Vielä hetki julkaisuun, mutta kannatti tehdä.

Äänien maisema

Pelin musiikin on säveltänyt englantilainen kaksikko The Audio Grill, jonka jäsenet David ja Mike innostuivat välittömästi pelistä nähtyään loppukesästä pelin demoversion. Sovimme, että he säveltävät peliin syntetisaattoreiden sävyttämän, tunneskaalaltaan miellyttävästä painostavaan etenevän ääniraidan. Korvauksena The Audio Grill saisi 10 prosenttia pelin ensimmäisen neljän kuukauden myynneistä. Ohjeeksi sävellystyöhön he saivat vain löyhän heiton: ”Philip Glass kohtaa Giorgio Moroderin”. Kukin voi pelin musiikista toki itse päätellä, vastaako lopputulos toivetta.

Äänitehosteiden ensimmäiset versiot kaivettiin ilmaisista lähteistä ja osittain myös luotiin itse GoldWave-äänieditorissa. Julkaisuversioon The Audio Grill nauhoitti osan äänistä uudelleen, jotta ne sopisivat paremmin musiikkiin.

Tärkeänä viimeisenä silauksena Division Cellin äänimaisemaan toimii Announceriksi nimetty naisääni. Ääni kommentoi valikoissa liikkumista, pelisuoritusta tasojen väleissä sekä Time Attack -tilan pelitapahtumia. Käytimme Philippa N:n ääninäyttelijän lahjoja myös pelin trailerivideossa. Hän teki kaikki nauhoitukset The Audio Grillin studiolla Miken ohjaamana.

Äänisuunnittelu mobiilipeleihin on omalla tavallaan kiittämätön työ. Monet pelaavat pelejä puhelimellaan tai tabletillaan julkisilla paikoilla, ja siten äänien päälläolo voidaan kokea häiritseväksi ja kiusalliseksi. Monet vaivalla tehdyt kappaleet ja muut äänet jäävät kokonaan useilta pelaajilta kokematta.

Sopii toivoa, että Division Cellin pelaajat rukkaavat äänenvoimakkuuden ylös. Musiikki muodostaa jopa 70 prosenttia koko pelipaketin koosta. Tämä on pelin ainoa osa-alue, jossa ei pyritty kotekohkuuteen tai minimalismiin vaan annettiin täysin vapaat kädet musiikin luojille. Äänimaisema on äärimmäisen tärkeä vaikutin osana pelikokemusta. Se luo pelaajalle tunnekokemuksia ja antaa pelille oman persoonallisen leimansa.

Itsenäinen peli – oma hinnoittelupolitiikka

Tarjosimme peliämme aluksi monelle taholle, esimerkiksi Roviolle. Kukaan ei kuitenkaan osoittanut kiinnostusta, joten päätimme perustaa oman osakeyhtiön ja pelistudion. Se voisi myös toimia alustana myöhemmin tuleville peleille ja niiden kehitykselle.

Skrollissakin vaikuttanut Jari ”Jaffa” Jaanto – yksi IRC-Gallerian ja Ninchatin perustajista – lähti pelin viimeisinä kehityskuukausina perustajajäseneksi ja kokeneeksi neuvojaksi mukaan uuteen Hyperspace Yard -studioomme. Näin voimme julkaista pelin juuri siten kuin tahdomme, eikä sen sisältöön tarvitse tehdä kyseenalaisia rahanahneita kompromisseja.

Uskomme, että pelimme on puolentoista euron arvoinen molemmilla mobiilikäyttöjärjestelmillä. Sen satunnaisgeneroituihin kenttiin perustuvat pelimuodot tarjoavat potentiaalisesti rajattoman uudelleenpeluuarvon kertaostoksella.

Julkaisua miettiessä ja suunnitellessa tulee pakostakin tietoiseksi, miten helposti tälläkin pelillä olisi voinut rahastaa Freemium-mallin puitteissa. Pelissä voisi myydä vinkkikolikoita ratkaisua varten tai lisää tasopaketteja, jotka peli voisi jo nytkin luoda muutaman rivin koodilla.

Emme lähteneet rahastusjunaan. Johtava ajatus peliä tehdessä oli, että sen pitää olla sellainen, jota haluamme itsekin pelata julkaisun jälkeen ja jossa ei voi ostaa tietään voittoon. Luotamme siihen, että pelaajat ovat valmiita maksamaan laadukkaasta pelistä, jos se on markkinoitu oikein.

Tietoa pelistä jaetaan Facebookissa, Twitterissä ja muissa sosiaalisissa medioissa. Mainontapalveluina käytämme aluksi Google Adwordsia ja lisäksi kokeilemme varmasti myös muita reittejä tuoda peliä ihmisten tietoisuuteen. Olemme myös ottaneet huomioon pelin sisäiset tavat levittää sanaa. Verkossa pelattavat ja helposti jaettavat kentät mahtuvat kokonaisuudessaan yhteen internetlinkkiin, joka on helppo jakaa esimerkiksi tekstiviestillä sekä ircin tai muiden pikaviestimien välityksellä. Tämä tekee pelikokemuksen jakamisesta äärimmäisen helppoa.

Pelin kotisivut ja traileri löytyvät osoitteesta http://cell.hyperspaceyard.com.

Oma vauva on kaunein

Pimiä toimitti meille myös lyhyen kuvailevan tekstin itse pelistä. Tämä ei ole puolueeton peliarvostelu, vaan kehittäjän oma näkemys. Ottakaa itse selvää, onko se turhaa kehua vai vaatimatonta vähättelyä:

”Division Cell on pelkistetysti esitetty palapeli. Division Cell on tarina kuvioista. Division Cell on hitaasti aukeava origami ja muodon eleganttia harmoniaa. Sen musiikissa kohtaavat FM-synteesin muisto ja elektropopin hyökkäävät äänimaisemat. Se on jotain niin yksinkertaista ja siltikin kätkee sisäänsä loputtoman monimutkaisen paljon niin vähässä.”

Ei näin!

Paina nappia, katso videota

Alussa maa oli autio ja tyhjä, ja videopelien grafiikka koostui legoukoista. Sellaisenaankin pelit kelpasivat monille, mutta niiden tekijät halusivat enemmän: elokuvatasoista kuvaa, jota voisi pelata. Ainakin tavallaan.

Teksti: Mikko Heinonen Kuva: Ville-Veikko Heinonen

Peliautomaattien eläessä 80-luvun alussa kultakauttaan käytiin kovaa kamppailua pelaajien kolikoista. Uusia innovaatioita esiteltiin nopeaan tahtiin. Mustavalkoisia hahmoja väritettiin ensin kalvoilla, joita liimattiin näyttöön. Sitten saatiin jo värigrafiikkaakin. Edelleen ruudulla kuitenkin vilisi yksinkertaisia hahmoja, koska laitteisto ei muuta osannut tuottaa.

Niiden rinnalla Dragon's Lair vuodelta 1983 näytti uskomattoman hienolta. Tikku-ukkojen sijaan ruudulla seikkaili animaattori Don Bluthin upeasti piirtämä sankari, joka pelasti prinsessaa linnasta. Sujauttamalla kolikon koneeseen sai ikään kuin pelata piirroselokuvan. Miten tämä oli edes mahdollista?

No, eihän se oikeasti ollutkaan. Dragon's Lair koostui LaserDisc-kuvalevystä, johon koko animaatio oli tallennettu pätkinä. Pelaaja ohjasi käytännössä sitä, toistettiinko seuraavaksi tarinalle jatkoa vai jokin näyttävä kuolema. Useimmiten jälkimmäinen, sillä peli oli vain reagoimista ruudulla välähtäviin esineisiin. Niiden mukaan piti osata ohjata oikeaan suuntaan tai painaa nappia oikealla hetkellä.

Peleinä tämäntyyppiset LaserDisc-automaatit olivat tietenkin köykäisiä kokemuksia. Oikean painallusten järjestyksen sai opeteltua, kunhan jaksoi syytää riittävästi rahaa koneeseen. Kaunis grafiikka kuitenkin houkutteli pelaajia kuin rihkama variksia, ja LaserDisc-pelejä ilmestyi muitakin. Hieman mielekkäämpiä niistä olivat sellaiset, joissa kuvalevy sisälsi vain näyttävää taustagrafiikkaa.

> *Pelaaja ohjasi sitä, toistettiinko tarinalle jatkoa vai näyttävä kuolema.*

Tauti leviää koteihin

Videoreaktiotestit eivät kuitenkaan malttaneet jatkaa vain pelihallien riemuna, vaan 80-luvun lopulla ne valmistelivat esiinmarssia koteihinkin. Lipunkantaja oli Control Vision -nimellä kulkenut pelikonsoli, jonka pelit tallennettiin VHS-kaseteille.

Control Visionin ajatus oli, että VHS sisältäisi sekä ohjelmadatan että videomateriaalin. Prototyypin pelimekaniikasta vastasi muutaman vuoden ikäinen ColecoVision-konsoli. Se kertoi osaltaan siitä, että tarkoitus oli vain lisätä videokuvaan yksinkertainen pisteidenlasku.

Leluyritys Hasbro lähti mukaan rahoittamaan hanketta. Jossain vaiheessa VHS-tekniikan rajat kuitenkin tulivat vastaan ja Control Vision kuivui kokoon. Sille ehdittiin kuvata kaksi kokonaista peliä: Night Trap, jossa suojeltiin teinityttöjä pahiksilta, ja Sewer Shark, jossa lenneltiin ennalta määritettyä reittiä videokuvan päällä ja ammuskeltiin. Molemmat oli toteutettu oikeilla näyttelijöillä, joiden suorituksia saattoi verrata lähinnä B-elokuviin.

Konsolin peruuntumisesta huolimatta ostava yleisö ei välttynyt näiltä taideteoksilta. Sega nimittäin esitteli 90-luvun alussa MegaDrive-konsoliinsa CD-aseman, joka pystyi toistamaan videokuvaa suoraan levyltä. Muuten teknisesti melko vaatimaton laite oli kuin tehty näitä "full motion video" -pelejä varten.

Night Trap ja Sewer Shark siirrettiinkin VHS:ltä CD-levylle, ja varsinkin ensin mainittu myi kohtalaisesti. Myynti selittyy pelin tyttölapsilla ja niiden herättämällä kohulla. Muita vastaavia seurasi pian, ja nappulan painelu seuraavan videopätkän näkemiseksi eli ja voi hyvin. Mega-CD:n ja aikalaistensa lisäksi videota katseltiin paljon myös kotitietokoneiden CD-ROM-peleissä.

Menneiden aikojen haamut

Vuosituhannen loppua kohti pelikoneiden teho kasvoi eikä erillisiä videojaksoja enää tarvittu. Ostava yleisökin kyllästyi suttuisiin ja huonosti näyteltyihin mukaelokuviin. Reaktiotestit melkein unohtuivat, kunnes Yu Suzuki esitteli Quick Time Eventiksi (QTE) kutsumansa pelimekaniikan osana mainiota Shenmue-seikkailupeliä.

QTE tarkoitti pelin keskelle upotettua toimintakohtausta, jossa pelaajan piti suorittaa tietty tapahtumasarja painamalla oikeaa nappulaa oikeaan aikaan. Käytännössä se oli kuin LaserDisc-peliä pelimoottorilla toteutettuna. Tämä uusvanha keksintö levisi pelisuunnittelijoiden keskuuteen kuin ruohikkopalo. Suzuki oli tarkoittanut QTE:n elävöittämään hidastempoista seikkailuaan, mutta pian kokonaisia pelejä tehtiin sen ympärille.

Kun vuonna 2013 käynnistät vaikkapa uuden Infinity Blade -pelin iPadillasi, saat nähdä upeaa grafiikkaa, jota täppärisi pystyy hädin tuskin edes tuottamaan. Pelaajan osa on piirrellä sormella erilaisia kuvioita ruutuun ja käynnistää ennalta määritettyjä iskusarjoja. LaserDiscin sijaan silmäkarkkia tuottaa nyt äärirajoilla toimiva grafiikkasuoritin, mutta älä anna sen hämätä: samasta asiasta tässä on kyse..

Mallin vikaa

Kokemuksia 3D-tulostamisesta

3D-tulostimet ovat monipuolinen työkalu, jolla voi valmiiden mallien monistamisen lisäksi räätälöidä juuri omiin tarpeisiinsa sopivia välineitä. Aivan mutkatonta uuden teknologian käyttö ei kuitenkaan ole.

Teksti: Annika Piiroinen Kuvat: Risto Mäki-Petäys, Flickr-käyttäjä edans

Risto Mäki-Petäys: ”Oli vuosi 2008, ja kuluttajakäyttöisiä 3D-tulostimia alkoi hiljalleen näkyä yhä enemmän Suomessa. Alternative Partyillä oli esillä eräs yksilö, mutta sitä ei ollut saatu silloin vielä toimintakuntoon. Ensikosketukseni 3D-tulostamiseen jätti minut janoamaan lisää tietoa.

3D-tulostimen nopeus ja tarkempi toiminta selvisi vasta, kun Helsinki Hacklab rakensi Torstin eli Prusa Mendel -mallisen, osittain kierrätysmateriaalista kootun tulostimen. Sen käyttöönotto oli yhtä salatiedettä, eivätkä odotukseni sitä kohtaan olleet kovin korkeat.

Muut hacklabilaiset olivat tulostaneet Torstilla lähinnä valmiita objekteja, mikä ei ollut minusta erityisen kiinnostavaa. Olen itse aina tulostanut sellaisia esineitä, joita ei saa valmiina eikä voi tehdä muulla tavalla. Kaikki ideat esineisiin ovat syntyneet käytännön tarpeesta. Lähtökohtana on ollut tarve ja ratkaisuna 3D-tulostaminen eikä toisinpäin. Ensimmäinen tarve, jonka ratkaisin 3D-tulostamalla, oli polkupyörän ohjaustankoon kiinnitettävä kännykkäteline.”

Navigaattori pyörään

”Halusin pyörääni telineen, johon voisin kiinnittää kännykän siten, että voisin käyttää sen navigaattorisovellusta ajaessani. Myytävät mallit oli joko tarkoitettu autoille tai väärän mallisille matkapuhelimille, joten itse tekeminen oli luonteva vaihtoehto.

Koska teline oli kookas, suunnittelin sen rakentuvan kolmesta osasta, jotka kiinnittyisivät ruuveilla yhteen. Tällä tavoin hukkamateriaalia syntyisi vähemmän, jos tulostettu osa osoittautuisikin vääränlaiseksi.

Ultimakerin ohjeet olivat epämääräiset. Tulostusjälki ei ollut yhtenäistä eikä tarttunut pintaan. Syy tosin selvisi vasta jälkikäteen: tulostuspäässä oli tukos. Torsti tulosti ensimmäiset kaksi osaa mallikkaasti, mutta kolmas osa hapertui jo tulostuksensa alkuvaiheessa. Hapertunut osa oli onneksi ainoa, jonka pystyi valmistamaan myös puusta, kun taas muiden osien tulostamatta jääminen olisi kaatanut koko projektin.

Aluksi hapertumisen syyksi epäiltiin, että asetuksissa ja ohjelmistoissa on tulostuskoneen vaihdon jäljiltä oltava jotain pielessä. Myöhemmin selvisi, että yksi tulostuspään lattakaapelin johdoista oli poikki. Katkeileva pursottimen signaali aiheutti reikäisyyttä, joka johti hapertumiseen. 3D-tulostukseen ryhtyvälle tällaiset yllätykset voivat tulla hyvin tutuiksi!

Tulostimen haluttomuus yhteystyöhön ei ollut ainoa ongelma. PLA-muovia tulostavaa hartsitulostinta käytettäessä painovoima ehtii vetää esinettä hieman kasaan ennen kuin käytetty muovi jäähtyy ja jähmettyy. Ruuveille tarkoitetut reiät turposivat umpeen, ja niitä piti suurentaa jälkeenpäin poralla. Poraamisen seurauksena osa kuitenkin lohkesi kulmasta. Mallia piti muuttaa siten, että reikä oli isompi eikä kasaan painuminen haitannut.”

> *”Hartsitulostinta käytettäessä painovoima ehtii vetää esinettä hieman kasaan ennen kuin käytetty muovi jäähtyy ja jähmettyy.”*

3D-mallista 3D-esineeksi

”3D-mallintaminen oli minulle tuttua jo demoskenestä ja graafikon hommista. Helsinki Hacklabilta minulle vinkattiin OpenSCAD-ohjelmisto, jolla voi suunnitella skriptattuja 3D-malleja. Sen käyttäminen ei vaatinut minulta muita uusia taitoja kuin ohjelmiston oman skriptikielen opettelemisen.

OpenSCADissa 3D-malli rakennetaan pelkästään ohjelmointikielen avulla eikä esimerkiksi piirtämällä. Se ei siis ole WYSIWYG-tyyppinen työkalu. Siinä voi määrittää, mitkä mitat riippuvat toisista mitoista, joten kun yhtä mittaa päivittää, päivittyvät kaikki siitä riippuvat mitat automaattisesti. Se tekee mallin muuttamisesta ja skaalaamisesta paljon helpompaa.

> *OpenSCADissa 3D-malli rakennetaan pelkästään ohjelmointikielen avulla.”*

OpenSCADilla voi tehdä myös moduuleja, esimerkiksi standardikokoisia ruuvinreikiä, joita voi sitten lisätä omiin malleihinsa tarvitsematta keksiä reikää joka kerta uudestaan. Tällöin voi helposti vaihtaa esimerkiksi ruuvikoon suuremmaksi.

Aloittelevalle 3D-printtaajalle suosittelen Trimble SketchUpia, joka on helpompi käyttää. Jos taas on mallinnuskokemusta eikä pelkää skriptaamista, voi OpenSCADillakin lähteä suoraan leikkimään. Moni ei enää suosittele OpenSCADia, mutta itse en ole löytänyt toista ohjelmaa, joka olisi niin näppärä ja tekisi sen, mitä haluan.”

Suunnittelija mokaa

”Kesällä 2011 polkupyöräni lukkoteline hajosi ajaessani katukiveykseen, ja uusikin teline hajosi samoissa olosuhteissa pian käyttöönoton jälkeen. Valmistajan malli ei selvästikään kestänyt katuajoa.

Alkuperäisessä mallissa oli jousimekanismi, jota ei pystynyt 3D-tulostamaan, joten jouduin suunnittelemaan lukkotelineen kokonaan itse.

Otin mitat pyörän satulatolpasta sekä lukosta ja loin mallin, joka kestäisi tärinää ja törmäyksiä. Katsoin tarkkaan, että lukko mahtuisi telineeseen osumatta satulaan tai renkaaseen. Tein myös telineestä riittävän paksun, jotta ruuvinreikien laajentaminen poralla ei aiheuttaisi lohkeamia muoviin.

Tulostaminen meni ensimmäisellä kerralla nappiin, ja teline kiinnittyi satulatolppaan juuri oikealla tavalla. Lukkokin mahtui telineeseen eikä osunut satulaan. Oli vain yksi pieni ongelma: itse lukon asettaminen telineeseen oli mahdotonta. Koko satula olisi pitänyt irrottaa, jotta lukon saisi pois ja takaisin. Pikku vikoja… Ajan nykyään ilman lukkotelinettä.”

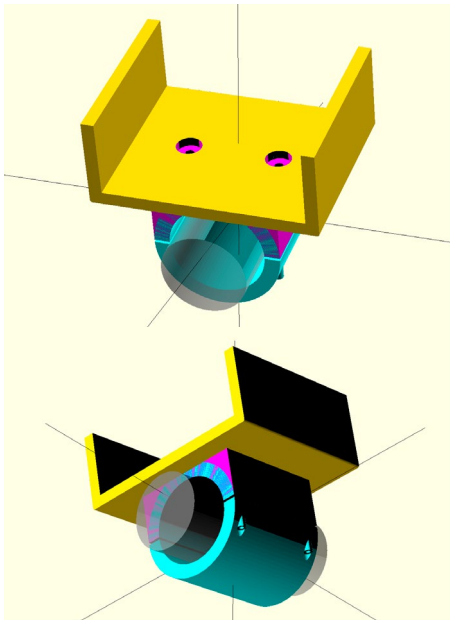
Parempi kuin uusi

”Aina ei tarvitse suunnitella esinettä tyhjästä, vaan joskus voi käyttää olemassa olevaa esinettä mallina. Tällöin virheitä sattuu lähinnä tulostustekniikkaan liittyvissä asioissa, kuten kasaan painumisten ja valumisten vuoksi.

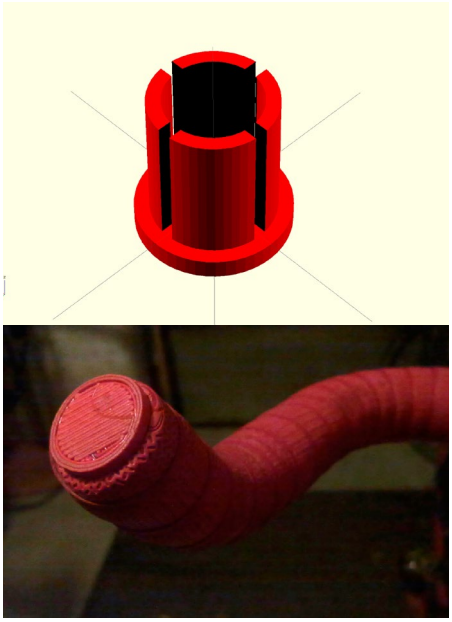
Pyöräni ohjaustangon päässä olevat tulpat irtosivat jatkuvasti, joten päätin tulostaa itse pidempivartisen mallin. Se osoittautui niin hyväksi, että se pysyi irtoamatta ja ehjänä, kunnes ajoin kolarin joitakin vuosia myöhemmin. Tulpan suunnittelu oli hyvä esimerkki siitä, mitä 3D-tulostaminen voi parhaimmillaan olla: pääsee itse parantelemaan esineen muotoilua käytännössä.

> *Onnistunut käyttöesine on sellainen, jota ei tarvitse ajatella.”*

Onnistunut käyttöesine on sellainen, jonka olemassaoloa ei juuri kukaan huomaa tai jota ei tarvitse ajatella. Se toimii niin täydellisesti. On hieno tunne, kun on onnistunut suunnittelemaan jotain paremmin kuin henkilö, joka tekee sitä työkseen.”



Kännykkätelineen 3d-malli, jossa väritettynä eri värillä kukin kolmessa osassa tulostuva osa sekä pyörän stemmi.



Ohjaustangon tulppa 3d-mallina ja tulostettuna lopullisessa kohteessaan.

Muistilista PLA-muovin tulostajalle

[yksinkertaiset mustavalkoiset kuvat valumisjuttuihin?]

1. Ota selvää tulostimen rajoituksista, esimerkiksi tulostetun esineen maksimimitoista.
2. Yli muutaman sentin pituiset, tyhjän päällä makaavat ”siltarakenteet” eivät sovellu tulostamiseen.
3. Jos tulostettavan pinnan alla on tyhjää, pinta painuu jonkin verran koveraksi. Suorat pinnat kannattaa tulostaa pystysuoraan tai tulostusalustaa vasten. Painumista voi jonkin verran vähentää säätämällä tulostusohjelma rakentamaan tukirakennetta tulosteen alle.
4. Samasta syystä reiät painuvat kasaan, jos niiden akseli on vaakasuora. Esine kannattaa tulostaa asennossa, jossa reiät ovat kohtisuoraan tulostusalustaa vasten.
5. Jos esineessä on useampia reikiä eikä kaikkia saa tulostuksessa pystyasentoon, kannattaa harkita esineen tulostamista useammassa osassa.
6. Tulostimen säädöt eivät ole aina täydellisesti kunnossa, minkä takia kappaletta voi joutua korjaamaan jälkeenpäin esimerkiksi viilaamalla tai mattoveitsellä.
7. Suunnittele malli huolellisesti, ota huomioon kaikki tarvittava ja varaudu yllätyksiin. Tulostuspää saattaa vaikka tarttua kiinni tulostettavaan esineeseen ja eksyä koordinaateista.
8. Mieti haluamasi tarkkuus (onko puolen millimetrin heitto ongelma) ja ota se huomioon suunnittelussa. On myös muistettava, että tulostimen tarkkuus on rajallinen.
9. Varaudu siihen, että tulostimella on huono päivä.
10. Korjaa mallia testitulosten perusteella.

Selvää säästöä

”Olin myös käyttämässä Suomen ensimmäistä sarjavalmistettua 3D-tulostinta, MiniFactorya, kun sellainen saatiin MikroBitin tiloihin testattavaksi.

Silloin paikalla ollut tietokirjailija ja tietotekniikka-asiantuntija Petteri Järvinen tuli tiirailemaan olkani yli projektiani eikä tahtonut millään uskoa, että 3D-tulostamisessa on mitään järkeä. Hän epäili, että kallis tulostin maksaisi harvoin itseään takaisin ja että tuotteen ostaminen kaupasta olisi paljon järkevämpää. Yhteiskäyttöiset tulostimet kirjastoissa ja hackerspaceissa saivat kuitenkin hänen mielensä muuttumaan suopeammaksi. Osasin myös antaa esimerkkejä tulostusprojekteista, jotka säästivät rahaa.

Kun lakkautettua mallia oleva vanha pistorasiakotelo hajosi makuuhuoneestani, vaihtoehtona olisi ollut pulittaa sadoista tuhanteen euroa sähkömiehelle pistorasioiden uusimisesta. Samalla tavalla rahaa säästyi, kun tuttuni vuokraaman asunnon vanhasta uunista murtui luukun kahvan bakeliittinen kannatin. Olisi pitänyt hankkia kokonaan uusi uuni, mikäli mitoiltaan sopivaa mallia olisi edes löytynyt. Turhaa askarteluahan tämä 3D-tulostaminen siis on.”

Kuumat paikat

”Uunin luukun kahvan kannattimen korvaaminen oli moneltakin kannalta mielenkiintoinen projekti. Kannattimeen meni uunin luukun läpi metallisia pultteja, joten kuumuus piti ottaa huomioon. Puu olisi ollut luonteva materiaalivaihtoehto, mutta osa oli sen mallinen, että sitä on mahdotonta työstää puusta.

Hacklabin Ultimaker tulosti vain PLA-muovia, joka pehmenee liian alhaisessa lämpötilassa eikä sovi yhteen uunin kuumuuden kanssa. Testasin kuitenkin tulostamani osan kestävyyttä lämmittämällä uunia täydellä teholla puoli tuntia, eikä pehmenemistä ilmennyt. Ilmeisesti metalliosat eivät johtaneet lämpöä liikaa läpi.

Vuokralainen ei ole valittanut tähänkään mennessä uunista, joten uskoisin, että ongelmia ei ole ollut.

> *Vuokralainen ei ole valittanut tähänkään mennessä uunista.”*

Kannattimen mallintaminen ei ollut helppoa, koska sen sisällä oli neliskulmaisia, hyvin täsmällisiä muotoja, viistoja tasoja sekä onkaloita. Siinä oli myös epämääräisesti kaartuvia linjoja, jotka piti hienosäätää kohdilleen muutaman testitulostuksen avulla.

Toinen kahvan kannatin oli ehjä, ja sain siitä tavallisen tasoskannerin ja piirto-ohjelman avulla laskettua kaltevuuskulmat ja tangon ääriviivat. Onkaloiden syvyyden mittaamiseen käytin työntömittaa.

Koko projektiin meni kolme iltaa, mutta kun esineen kaarevuus oli saatu säädettyä testipalojen avulla kohdalleen, kokonainen esine toimi lopulta ensi yrittämällä.”

Klaustrofoobikot hississä

”Olen toteuttanut useita projekteja vuosien varrella, mutta en aio käydä niitä kaikkia tässä läpi. Haluan kuitenkin kertoa vielä eräästä projektista, joka ei vielä ole onnistunut mutta jonka taustalla on aito tarve. Kerrostaloni hissiin nimittäin juuttuu jatkuvasti ihmisiä.

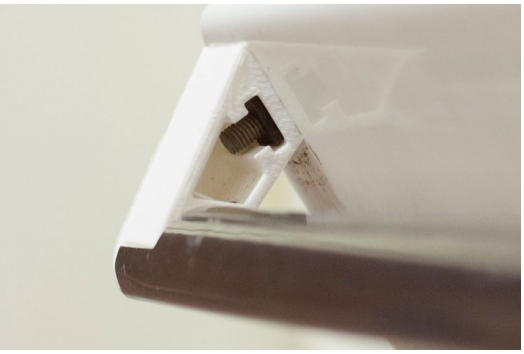
Olin eräänä yönä nukkumassa, kun naapurini soitti ovikelloa. Hänen klaustrofobinen ystävänsä oli juuttunut hissiin, ja hän oli saanut jostain päähänsä, että minä olisin oikea ihminen auttamaan. Hissimekaanikko oli hälytetty, mutta häntä ei kuulunut. Hississä jumissa ollut mies jo hyperventiloi paniikista ja takoi ovea.

Periaatteessa hissin ovea ei pitäisi saada auki siinä tilanteessa lainkaan, mutta se ei tietenkään estänyt kokeilemasta. Onnistuin lopulta kikkailemaan lukon auki kuusiokoloavaimella. Ahdistunut mutta huojentunut pettäneen tekniikan uhri kömpi ulos hissistä. Hän ei ole sen koommin käyttänyt talomme hissiä ja halaa aina onnellisesti, kun satumme törmäämään.

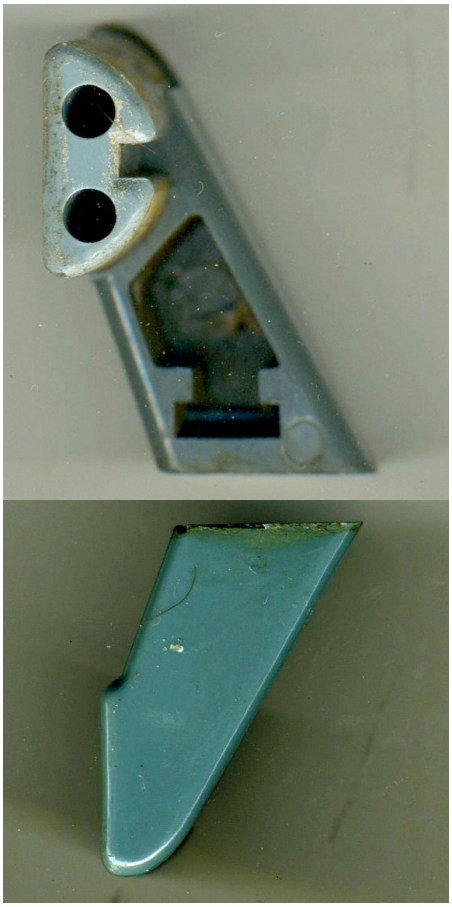
Älä kokeile tätä kotona. Hissikuiluun voi pudota, jos oven avaa korin ollessa kerrosten välissä.

Kyseinen tapaus ei suinkaan ollut ainut kerta, kun joku jumittui hissiimme – sitä tapahtui vuosittain. Kuusiokoloavaimella lukon sörkkiminen ei pidemmän päälle tee lukolle hyvää, joten tarvittiin pysyvämpi ratkaisu. Lukon avaamiseen tarkoitettu työkalu on turvallisuussyistä lähinnä hissimekaanikkojen saatavilla, joten päätin kokeilla oman tulostamista.

Vanhat, likaiset ja haljenneet palat ennen operaatiota.



Uusi, siisti 3d-tulostettu pala lopullisessa kohteessaan.



Tasoskannerilla otetut mittakuvat alkuperäisestä osasta.

Projektit 2011-2013
(X: ei päätynyt tuotantoon)

Projekti	Tulos
Kännykkäteline polkupyörään	:)
Polkupyörän lukkoteline	:I
IDE64:n kotelo	X
Polkupyörän ohjaustangon tulppa	:D
Elektroniikkapelin paristokotelon kansi	:)
Pistorasian kansi	:D
Hissiavain	:(
Lasten älypelilelun osa	X
Hellan kahvan kannatin	:D
Oman sekvensserin adapterin palasia	:I
USB-reiän suojatulppa	:D
Polkupyörän vaihteiston kansi	X
Kitaran hihnan tappi keytariin	:D

Lukon muoto oli kuitenkin niin epäsäännöllinen, ja avaimeen olisi tarvinnut niin ohuita kohtia, ettei yksikään koekappale onnistunut. Luovutin toistaiseksi.”

Hae mallit netistä

Projekteja varten laaditut 3D-mallit ja apuvälinemoduulit löytyvät Skrollin sivuilta, osoitteesta http://skrolli.fi/2013.4. Moduuleihin sisältyy muun muassa M-standardikokoisten pulttien, mutterien sekä vastaavat reikien moduulit ja viivoitinmoduuli, jonka avulla heikompikin matikkapää voi tarkistaa mittaukset. Sivustoilta kuten Thingverse (http://www.thingiverse.com/) löytyy muiden tekemiä valmiita malleja, joiden tulostaminen voi olla helpoin tapa aloittaa 3D-tulostuskokeilut.

Minustako 3D-tulostajaksi?

Yleisimmät ja edullisimmat tulostimet perustuvat fused deposition modelling -tekniikkaan. Niissä muovifilamenttia syötetään kuumaan tulostuspäähän. Sula muovinauha työntyy ulos tulostuspään ohuesta kärjestä ja jähmettyy paikalleen. Tulostuspää ja -alusta liikkuvat kolmella akselilla toistensa suhteen piirtäen langalla kappaleen muodon alhaalta ylöspäin, yksi kerros kerrallaan. Kerroksen valmistuttua pää siirtyy hieman ylös ja uusi siivu tulostetaan. Esine kasvaa hitaasti alustasta.

Perustulostin, jossa on yksi tulostuspää, maksaa hieman alle tuhannesta kolmeen tuhanteen euroa. Kaksisuuttimiset mallit ovat kalliimpia, ja niillä voi käyttää kahta eri väriä tai materiaalia samassa kappaleessa. Rakennussarjana monet tulostinmallit saa edullisemmin.

Filamenttitulostimet käyttävät yleensä PLA- tai ABS-muovia. PLA-muovi on kestävä maissipohjainen, kierrätettävä ja biohajoava muovi. ABS-muovia tulostettaessa tarvitaan esilämmitettävää alustaa, joka on hankittava erikseen moneen tulostimeen. ABS:ää käytetään muun muassa Legoissa.

Muovifilamenttitulostinta pääsee kokeilemaan esimerkiksi eri paikkakuntien hackerspaceissa ja pääkaupunkiseudulla myös Tapiolan kirjastossa, Lasipalatsin Kohtaamispaikassa ja Aalto FabLabin tiloissa.

Monia muitakin tulostintyyppejä on olemassa. Stereolitografiassa kovetetaan nestemäisiä fotopolymeerejä valolle altistamalla. Sintrauslaitteet taas sulattavat pientä muovi- tai metallijauhetta yhteen kerros kerrokselta. Laajaa valikoimaa jauhemaisia materiaaleja voidaan kovettaa myös nestemäisillä sidosaineilla, joita tulostetaan jauhekerroksen päälle. Suurin osa muihin tulostustekniikoihin pystyvistä laitteista on toistaiseksi vähintään kymmenien tuhansien eurojen hintaisia. Yliopistojen laboratorioissa voi olla jopa miljoonien eurojen ultraviolettifemtosekuntilasertulostimia tai muita villejä viritelmiä.

Jauhetulostusmenetelmä on kuluttajan käytettävissä erilaisten palveluiden kuten Shapewaysin kautta. Palveluun ladataan 3D-malli ja valitaan haluttu materiaali, ja tulostettu esine saapuu postissa muutaman päivän kuluttua. Mitä isompia määriä tulostaa, sitä halvemmaksi esineen kappalehinta tulee. Tällä tavalla esineiden tekeminen on kuitenkin hitaampaa ja kalliimpaa kuin esimerkiksi yleisessä käytössä olevilla muovifilamenttitulostimilla.

- Shapeways: http://www.shapeways.com
- Tapiolan kirjasto: http://www.helmet.fi/fi-FI/Kirjastot_ja_palvelut/Tapiolan_kirjasto
- Lasipalatsin Kohtaamispaikka: http://www.lasipalatsi.fi/kohtaamispaikka
- Aalto Fablab: http://fablab.aalto.fi
- Hacklabeja Suomessa: http://hackerspaces.org/wiki/Finland

OpenSCAD

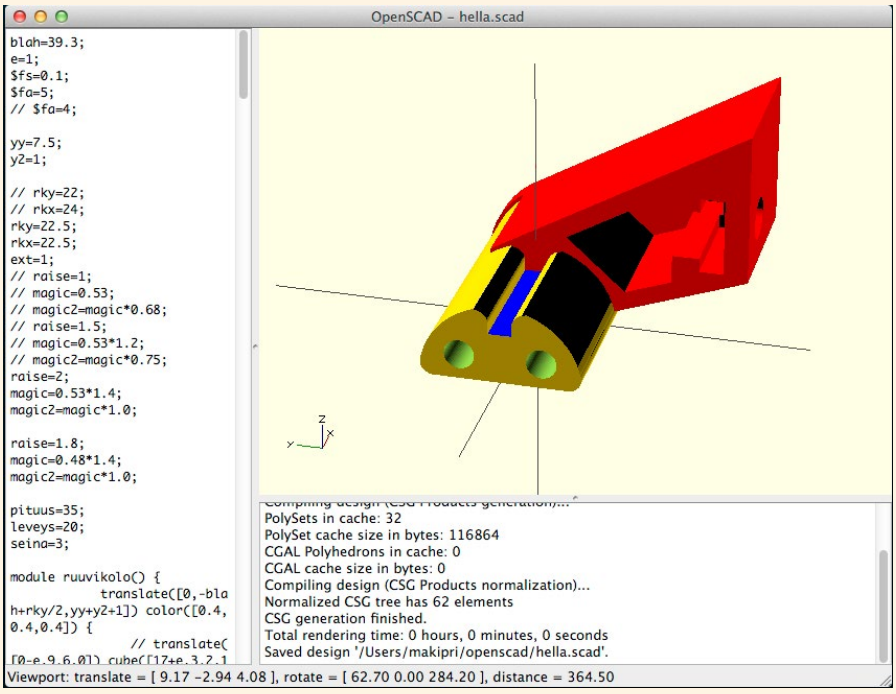
Risto Mäki-Petäys

OpenSCAD on CGAL-kirjastoon pohjautuva vapaa ohjelmisto kiinteitten kolmiulotteisten CAD-mallien suunnittelemiseen. Se ei sisällä graafisia työkaluja esineen mallintamiseen ja muokkaamiseen, vaan mallit luodaan skriptikielellä. Tämä mahdollistaa muun muassa aiemmin suunniteltujen osien helpon ja tehokkaan jälleenkäytön, mallien parametrisoinnin sekä toistuvien toimenpiteitten näppärän automatisoinnin.

Mallit voidaan luoda kasaamalla erilaisia primitiivejä (kuutio, sylinteri, pallo) yhteen, vähentämällä niitä toisistaan boolen algebralla tai vaihtoehtoisesti luomalla ne useammasta monikulmiosta. Myös ohjelman ulkopuolelta tuotuja, DXF-formaatissa olevia malleja voidaan hyödyntää. Mallin eri osia voi havainnollistamisen vuoksi pinnoittaa haluamillaan väreillä tai säätää läpikuultaviksi.

OpenSCAD on näppärä pieni ohjelma, joka on helppo ottaa käyttöön. Vaikka oletettaisi mallin kirjoittamisen olevan työlästä skriptikielellä, voi OpenGL-esikatselun avulla kuitenkin nopeasti hahmottaa kappaleessa tapahtuvat muutokset. Joissakin tapauksissa monimutkaisemmalla boolen algebralla rakentuvissa esineissä saattaa esikatselun piirrossa ilmetä bugeja. Lisäksi joillakin näytönohjaimilla ilmenee piirrossa puutteita. Vapaaehtoisesti kehitetyssä ohjelmassa saattaa mennä pidemmänkin aikaa reagointiin, ellei itse kyllästy ja päätä korjata lähdekoodia.

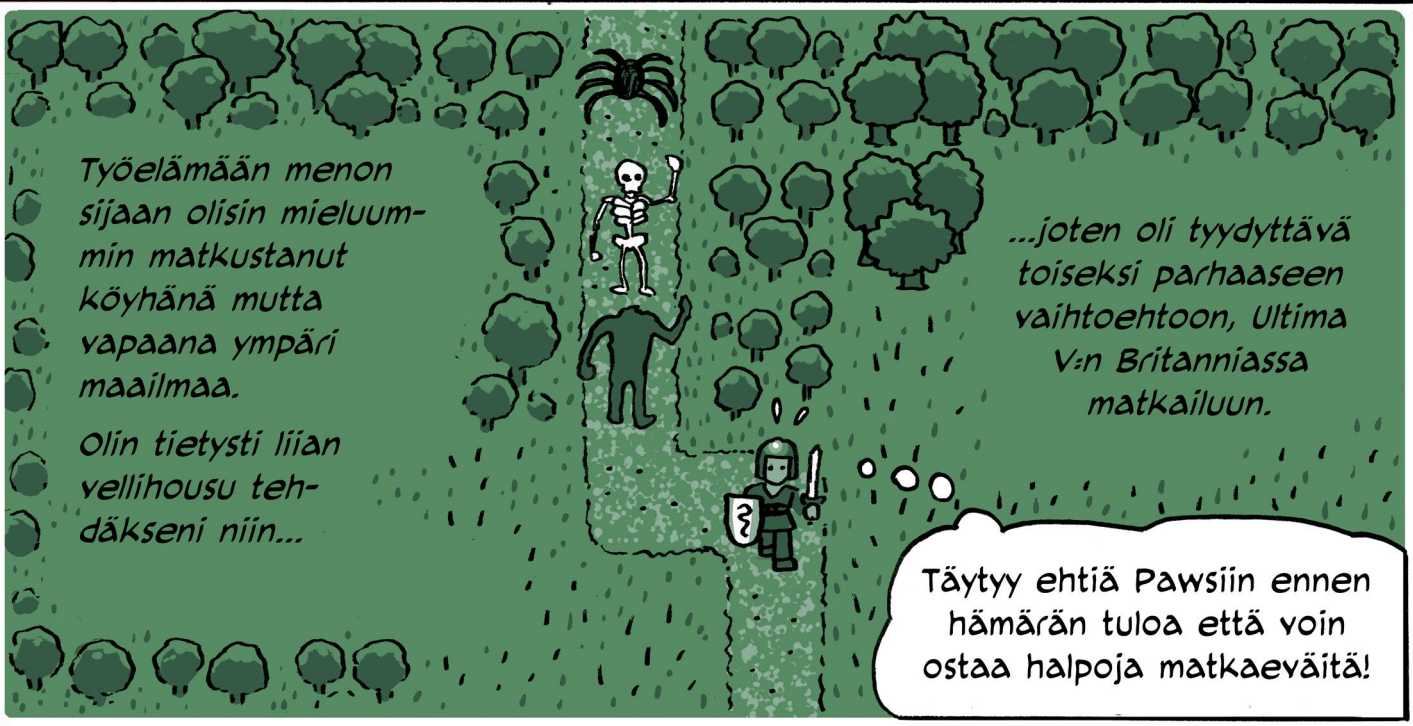
http://www.openscad.org/

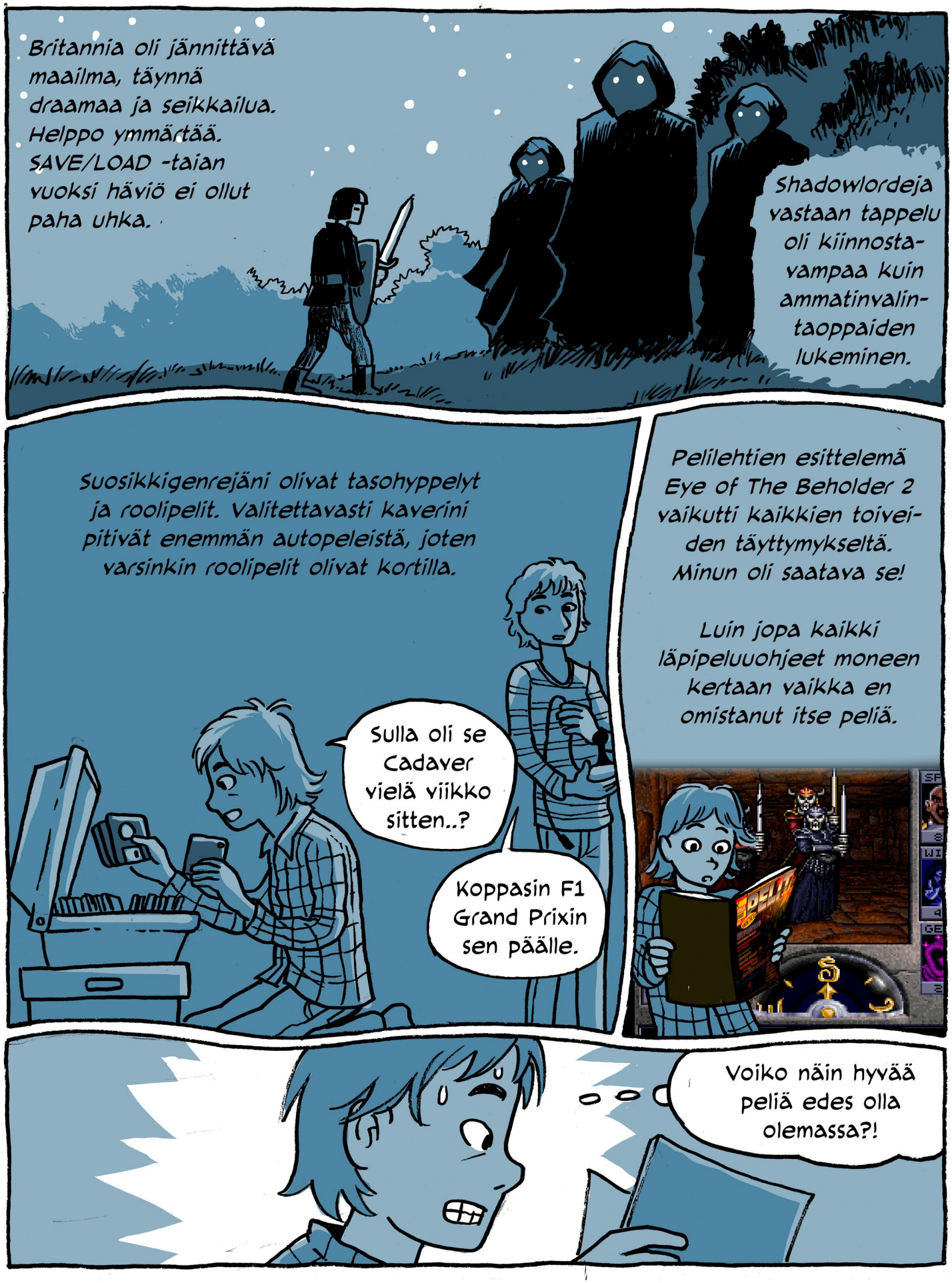


Nuoruuteni pelien parissa

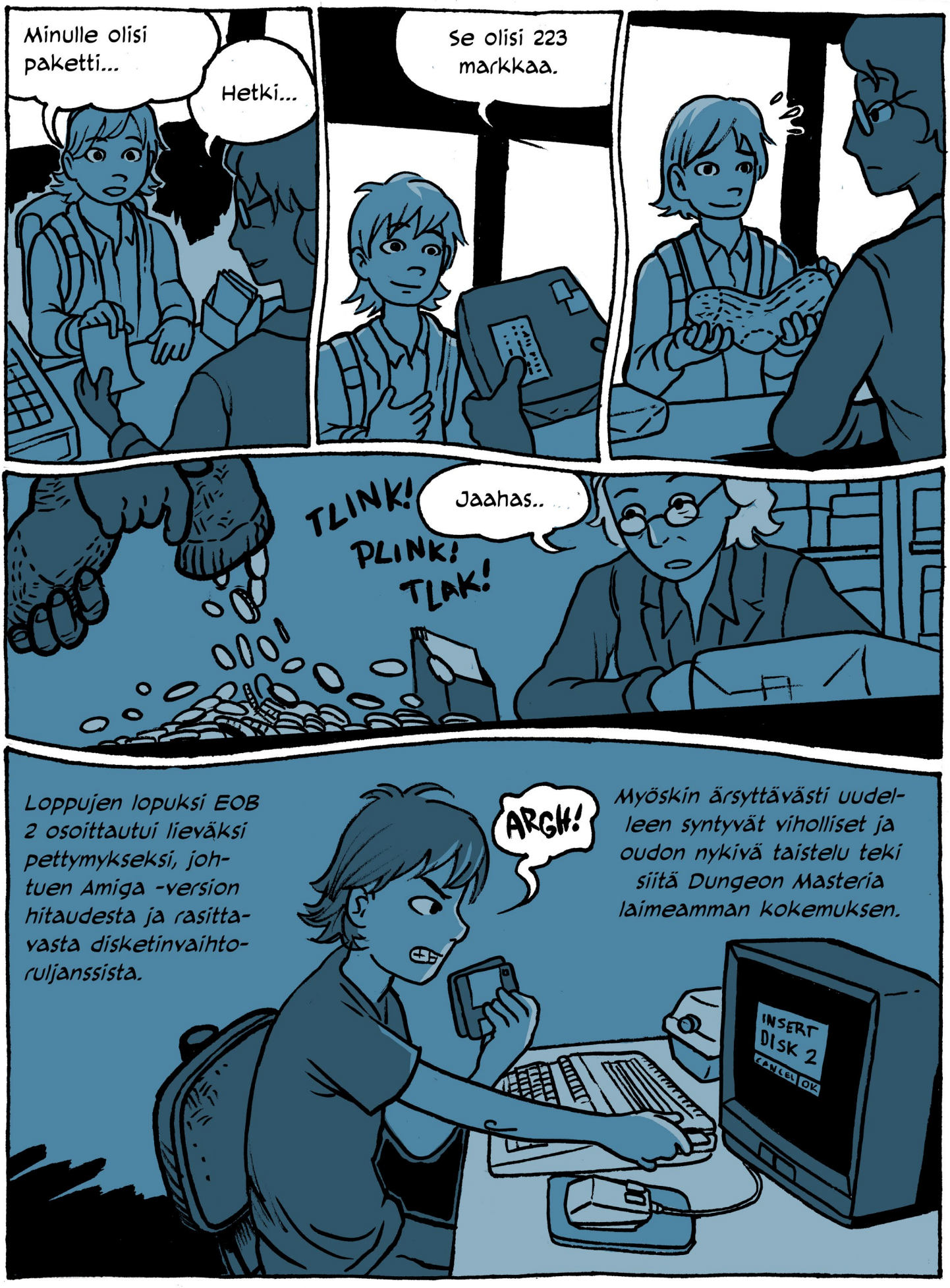

Mahtavia pelejä! Oi voi! Tämä on niin hienoa.
Joo. Se oli yksi nuoruuteni hienoimpia ja maagisimpia iltoja.


Minua ei kiinnostanut kaljanjuonti tai ostarilla hengailu..

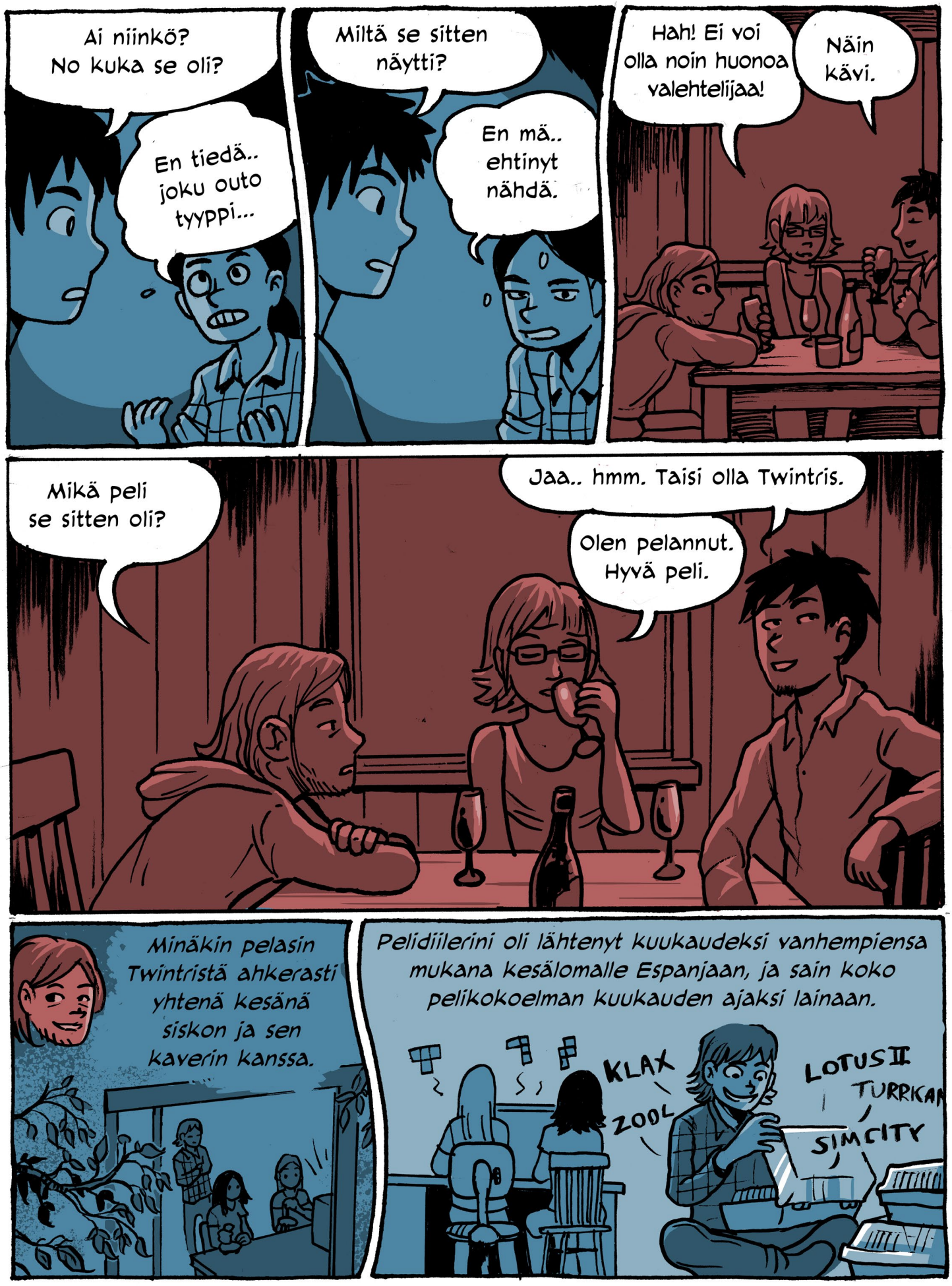

Työelämään menon sijaan olisin mieluummin matkustanut köyhänä mutta vapaana ympäri maailmaa.
Olin tietysti liian vellihousu tehdäkseni niin...
...joten oli tyydyttävä toiseksi parhaaseen vaihtoehtoon, Ultima V:n Britanniassa matkailuun.
Täytyy ehtiä Pawsiin ennen hämärän tuloa että voin ostaa halpoja matkaeväitä!

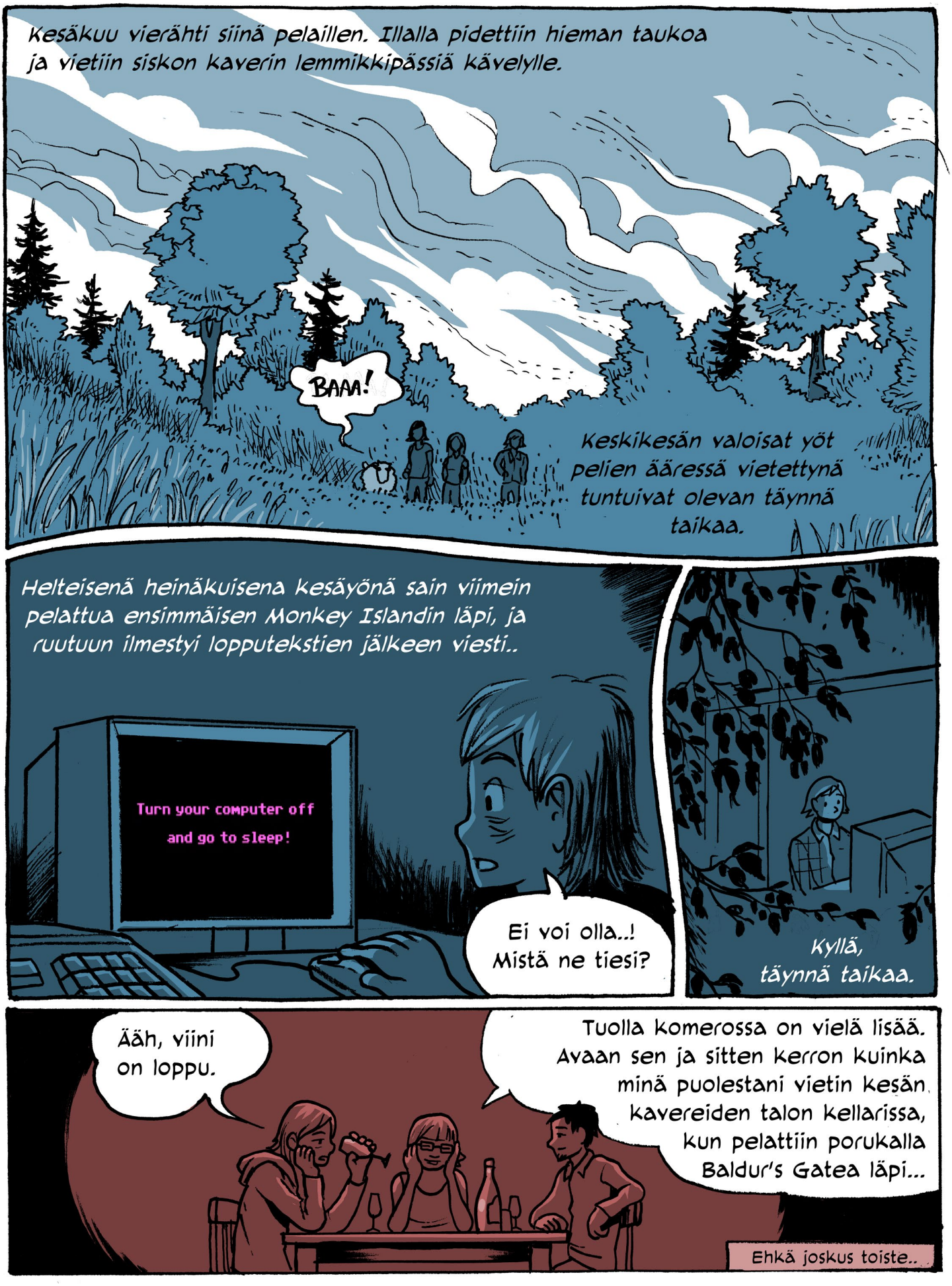

Britannia oli jännittävä maailma, täynnä draamaa ja seikkailua. Helppo ymmärtää. SAVE/LOAD -taian vuoksi häviö ei ollut paha uhka.
Shadowlordeja vastaan tappelu oli kiinnostavampaa kuin ammatinvalintaoppaiden lukeminen.
Suosikkigenrejäni olivat tasohyppelyt ja roolipelit. Valitettavasti kaverini pitivät enemmän autopeleistä, joten varsinkin roolipelit olivat kortilla.
Sulla oli se Cadaver vielä viikko sitten..?
Koppasin F1 Grand Prixin sen päälle.
Pelilehtien esittelemä Eye of The Beholder 2 vaikutti kaikkien toiveiden täyttymykseltä. Minun oli saatava se!
Luin jopa kaikki läpipeluuohjeet moneen kertaan vaikka en omistanut itse peliä.
Voiko näin hyvää peliä edes olla olemassa?!


Mielessäni EOB kakkosen Darkmoonin temppeli oli loputtomien mahdollisuuksien ja jännittävien seikkailujen paikka.
Monena aurinkoisena loputalven päivänä pyöräilin koulun jälkeen keskustaan.
Huh! Koulupäivä tuntui taas kestävän ikuisuuden.
AMIGA
C-64
OFFICIAL
COMPUTER PROD
235:-
BEHOLDER II
LEGEND OF DARKMOON
Viimein sain tarpeeksi viikkorahoja säästettyä voidakseni tilata pelin itselleni postimyynnistä, ja eräänä sohjoisena huhtikuisena iltapäivänä...
KLAK!


Minulle olisi paketti...
Hetki...
Se olisi 223 markkaa.
TLINK!
PLINK!
TLAK!
Jaahas..
Loppujen lopuksi EOB 2 osoittautui lieväksi pettymykseksi, johtuen Amiga -version hitaudesta ja rasittavasta disketinvaihtoruljanssista.
ARGH!
Myöskin ärsyttävästi uudelleen syntyvät viholliset ja oudon nykivä taistelu teki siitä Dungeon Masteria laimeamman kokemuksen.
INSERT DISK 2
CANCEL
OK


Sori pitkästä yksinpuhelusta, tämä aihe saa aina hieman herkistymään.
Kuullostaa ihan hyvin käytetyltä nuoruudelta.
Hah, muistanpa yhden jutun joka tapahtui minulle..
Olin yhden kaverin luona kylässä...
?
Mitäh?! Tämähän on..
...mun peli! Tunnistan jopa nämä naarmut laatikossa. Oon etsinyt tätä kaikkialta.
Miksi se on täällä?
No, öööö...
Ootko sä pöllinyt sen multa?
No en varmana ole! Oli.. joku tyyppi..
..joku tyyppi tuli pyörällä vastaan ja sanoi että ota tää peli!


Ai niinkö? No kuka se oli?
En tiedä.. joku outo tyyppi...
Miltä se sitten näytti?
En mä.. ehtinyt nähdä.
Hah! Ei voi olla noin huonoa valehtelijaa!
Näin kävi.
Mikä peli se sitten oli?
Jaa.. hmm. Taisi olla Twintris.
Olen pelannut. Hyvä peli.
Minäkin pelasin Twintristä ahkerasti yhtenä kesänä siskon ja sen kaverin kanssa.
Pelidiilerini oli lähtenyt kuukaudeksi vanhempiensa mukana kesälomalle Espanjaan, ja sain koko pelikokoelman kuukauden ajaksi lainaan.
KLAX
ZOOL
LOTUS II
TURRICAN
SIMCITY


Kesäkuu vierähti siinä pelaillen. Illalla pidettiin hieman taukoa ja vietiin siskon kaverin lemmikkipässiä kävelylle.
BAAA!
Keskikesän valoisat yöt pelien ääressä vietettynä tuntuivat olevan täynnä taikaa.
Helteisenä heinäkuisena kesäyönä sain viimein pelattua ensimmäisen Monkey Islandin läpi, ja ruutuun ilmestyi lopputekstien jälkeen viesti..
Turn your computer off and go to sleep!
Ei voi olla..! Mistä ne tiesi?
Kyllä, täynnä taikaa.
Ääh, viini on loppu.
Tuolla komerossa on vielä lisää. Avaan sen ja sitten kerron kuinka minä puolestani vietin kesän kavereiden talon kellarissa, kun pelattiin porukalla Baldur's Gatea läpi...
Ehkä joskus toiste..

Säätämisentäyteistä joulua kaikille Skrollin lukijoille!

T. Kapsi Internet-käyttäjät ry

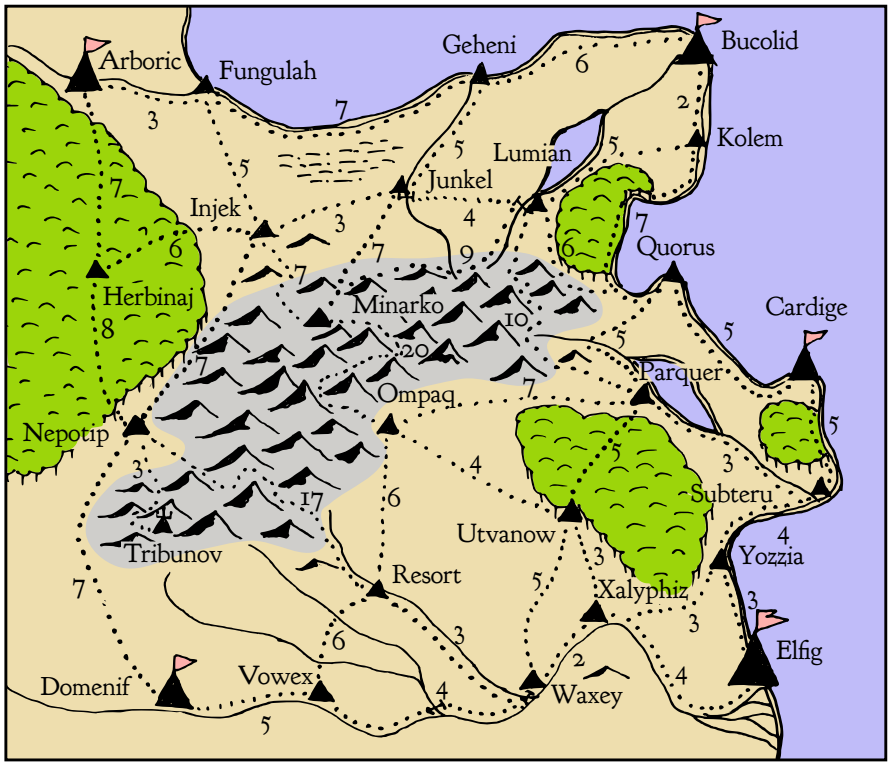
Pähkinänurkka

Tällä palstalla ilmestyy ohjelmointiin ja matematiikkaan liittyviä tehtäviä. Kaikki tehtävät on mahdollista ratkaista nopeasti sopivalla algoritmilla — ja osa ilman tietokonetta!

Lauri Alanko

Undigran valtakunnassa on levotonta. Maa on hajanainen, maasto on vaihtelevaa ja kaupungista toiseen on monen päivän matka. Kuningas Algor on juuri astunut valtaan. Auta häntä selviämään Undigraa koettelevista uhkista!

1. Arboricissa on kapina! Algorin täytyy päästä joukkoineen pääkaupunki Elfigistä Arboriciin mahdollisimman nopeasti ennen kuin kapina leviää. Mikä reitti hänen kannattaa valita?
2. Kapinaa johtanut velho saatiin vangittua, mutta kapina oli vain harhautus. Salaliitto Elfigissä yrittää kaapata vallan! Algorin täytyy palata nopeasti takaisin. Onneksi velho pystyy yhden kerran luomaan portin, joka siirtää Algorin välittömästi yhden etapin ylitse. Mikä reitti Algorin kannattaa nyt valita?
3. Algor ehtii viime hetkellä kotiin ja kukistaa salaliiton. Hän päättää olla jatkossa paremmin perillä valtakunnan asioista ja perustaa postiverkon tuomaan raportteja kaikista kaupungeista. Viestinviejiä tarvitaan yksi jokaista kaupunkien välistä päivämatkaa kohden, mutta Algorilla on vain sata viestinviejää. Minkä kaupunkien välille täytyy perustaa postilinjat, jotta verkosto ulottuisi sadalla viestinviejällä kaikkiin kaupunkeihin?
4. Nyt valtakunnan riesana ovat peikot, jotka tekevät tuhojaan syrjäseuduilla ennen kuin apu ehtii Elfigistä paikalle. Algor päättää perustaa kaksi tukikohtaa, joista sotilaat pääsevät nopeammin kaupunkien avuksi. Minne tukikohdat pitää perustaa, jotta kaukaisimpaankin kaupunkiin olisi mahdollisimman lyhyt matka jommastakummasta tukikohdasta?
5. Tukikohdista on apua, mutta peikot ehtivät yhä mellastaa liian pitkään. Algor määrää, että yksikään kaupunki ei saa olla yli kymmenen päivämatkan päässä tukikohdasta. Kuinka monta tukikohtaa vähintään tarvitaan, jotta tämä onnistuisi?
6. Peikot on lopulta saatu aisoihin ja valtakunnassa on taas rauha. Kuningas Algorin kruunajaiset lähestyvät. Undigrassa on tapana, että kuningas vannoo jokaisessa kaupungissa suojelevansa sen asukkaita. Sitten hänet kruunataan Tribunovin temppelissä auringonseisauksen aikaan. Auringonseisaukseen on enää 123 päivää! Kuinka Algor ehtii Elfigistä Tribunoviin ajoissa?



Undigra. Luvut ovat etappien matka-aikoja päivissä mitattuna. Kartta löytyy koneluettavassa muodossa osoitteesta http://skrolli.fi/2013.4/pahkina/

Kunnon työvälineet pelaamiseen - Cooler Master

CM Storm QuickFire TK, Cherry MX Red

QuickFire TK on taustavalaistu mekaaninen pelinäppäimistö punaisilla kytkimillä. Se on normaalia lyhyempi ja saanut pelaajien suosion ympäri maailman.

89,90 €

CM Storm QuickFire XT, Cherry MX Blue

QuickFire XT on mekaaninen pelinäppäimistö sinisillä kytkimillä. Se tarjoaa tukevan rakenteen, lasermerkityt näppäinhatut sekä mattapintaisen viimeistelyn.

79,90 €

CM Storm Ceres 500

Headsetin isot kuulokkeet ovat pehmeät ja poistavat häiritsevää taustaääntä. Niiden äänimaailma on suunniteltu pelaamiseen, joten saat selvän edun pelaamiseen PC:llä tai konsolilla.

64,90 €

CM Storm Havoc

CM Storm Havoc on 8200DPI Avago 9800 sensorilla varustettu pelihiiri. Se on suunniteltu tarkkaan pelaamiseen ja siinä on 8 täysin ohjelmoitavaa näppäintä.

49,90 €

Jimm's Pop Gamer

Jimm's Pop Gamer on yleisön suosikki. Tämän koneen naapurisikin ostaa, koska se tarjoaa parasta hinta/teho/laatu suhdetta. Nyt järjestelmälevynä SSD!

Intel i5-4670K 3.4GHz, 6Mt -prosessori | Intel Z87-piirisarjan emolevy | NVIDIA GeForce GTX 760, 2Gt GDDR5 -näytönohjain | 8Gt (2x4Gt) DDR3 1600MHz, CL9 -keskusmuisti | 120Gt, 2.5", Sata III -SSD asema | Cooler Master Silencio 550 Jimm's Edition -kotelo | Windows 7 Home Premium 64-bit

1069 €

JIMM'S
www.jimms.fi

Asiakaspalvelu
+358 29 70 70700
asiakaspalvelu@jimms.fi

Jimm'siltä löytyy nettitilauksiin aina toimituskuluton vaihtoehto!

Noutopalvelumyymälä Turku
Lukkosepänkatu 7
20320 Turku

Hinnat €/kpl, sis. alv24%